ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

PLUG-IN PARA LA ADMINISTRACIÓN DE RIESGOS SEGÚN CMMI NIVEL 2 EN PROYECTOS GESTIONADOS CON LA HERRAMIENTA REDMINE

David Ramírez Domínguez
José Javier Astráin Escola
Alberto Córdoba Izaguirre

Pamplona, 1 de julio de 2011

ÍNDICE

INTRODUCCIÓN

A lo largo del presente documento se ofrece la documentación sobre el Proyecto Fin de Carrera titulado "PLUG-IN PARA LA ADMINISTRACIÓN DE RIESGOS SEGÚN CMMI NIVEL 2 EN PROYECTOS GESTIONADOS CON LA HERRAMIENTA REDMINE"

Este proyecto ha sido realizado por David Ramírez en colaboración con la empresa D2D (Diseño y Desarrollo de Sistemas de Información). Dicho proyecto corresponde a la carrera de Ingeniería Técnica Informática de Gestión, impartida por la Universidad Pública de Navarra.

El documento ha sido dividido en varios capítulos atendiendo a las distintas fases presentes en el desarrollo del proyecto.

En los primeros capítulos, a modo de introducción, se analiza cuáles han sido las motivaciones que han llevado al desarrollo del mismo. En esta parte se muestra una visión general de sus objetivos, del entorno que lo rodea así como varios aspectos teóricos que a tener en cuenta.

Los siguientes capítulos van asociados al ciclo de desarrollo del software.

En la fase de inicio se realiza un estudio inicial de los requisitos del proyecto y de los posibles riesgos.

En la fase de análisis y diseño se describe detalladamente el modelo de datos así como las tablas que lo conforman y los diferentes casos de uso que encontramos.

Como parte final del documento, existe un capítulo dedicado a conclusiones y líneas futuras. Las conclusiones son tanto a nivel de realización del proyecto como de los resultados del producto desarrollado. En la segunda parte de este capítulo, se detallan las posibles mejoras que se pueden realizar y qué otras funcionalidades serían interesantes para mejorar el producto desarrollado. De esta forma, si en algún momento se desea dedicar más tiempo al proyecto, ya se conocerán cuáles son los puntos en los que es interesante dedicar el esfuerzo.

1.1 OBJETO DEL PFC

El proyecto a realizar consiste en el desarrollo de un *plugin*, para la versión de Redmine que el cliente tiene instalada en sus servidores y que tendrá como objetivo administrar los posibles riesgos que puedan surgir a lo largo del ciclo de vida de los proyectos, ayudándose para ello de un sencillo asistente que detectará dichos riesgos.

Redmine es una aplicación web desarrollada bajo el framework Ruby on Rails (RoR). Se trata de una herramienta multi-plataforma, de código abierto (open source) bajo licencia GPL, para la gestión de proyectos.

La ampliación objeto de este proyecto deberá afectar lo menos posible al núcleo original de la aplicación, por lo que, en tanto en cuanto sea posible, se hará efectiva en forma de plug-in. Servirá para gestionar los diferentes riesgos que puedan darse en un proyecto conforme a las metas y prácticas definidas por el modelo CMMI en el nivel 2.

Todo el desarrollo se ajustará a las convenciones impuestas por el *framework Ruby On Rails*, desarrollando en lenguaje de programación Ruby y siguiendo las políticas de programación de Redmine, como puede ser la internacionalización por medio de GLoc.

El proyecto será llevado a cabo dentro de la empresa *From Design To Development*, S.L. (D2D), dedicada al desarrollo de aplicaciones web. Este desarrollo le permitirá reducir el número de herramientas que necesite emplear para la gestión de proyectos según el modelo CMMI, reduciendo asimismo las necesidades de formación de sus equipos y los riesgos derivados de falta de sincronización entre herramientas (redundancia de trabajo, incoherencia de productos de trabajo y descentralización de información).

1.2 DESCRIPCIÓN DEL PFC

1.2.1 Marco de desarrollo

El proyecto se lleva a cabo dentro de una empresa dedicada al desarrollo de software.

D2D es una joven empresa de consultoría y desarrollo en el ámbito de las tecnologías de la información y comunicación (TIC). Tiene como misión dotar a sus clientes de herramientas versátiles e intuitivas, adaptadas completamente a sus necesidades y negocio, que faciliten su trabajo diario.

Radicada en Navarra y con el apoyo de CEIN (Centro Europeo de Empresas e Innovación de Navarra), D2D nació en marzo de 2007. La experiencia de sus fundadores en las últimas tecnologías orientadas al desarrollo web, forjada durante varios años trabajando en sectores tan exigentes como la banca o la industria de la automoción, dota de solidez al proyecto y, por consiguiente, a los desarrollos que se realizan.

D2D desarrolla aplicaciones web para una gran diversidad de empresas y sectores:

- eDependencia y eSalud
- Organismos públicos
- Energías renovables
- Industria de la automoción

Todos sus desarrollos tienden a englobarse en la Web 2.0. Esto supone, por ejemplo, la separación de contenido y diseño mediante el uso de hojas de estilo o el empleo de Ajax en páginas web dinámicas, constituyendo aplicaciones centralizadas que reemplazan a las habituales aplicaciones de escritorio.

D2D se planteó la obtención de la certificación CMMI ya que influye positivamente en la organización gracias a la mayor capacidad de control, predicción y visibilidad en la gestión, dirección, seguimiento y desarrollo de proyectos de software. Asimismo, permitirá competir cada vez mejor en los mercados nacional e internacional, asegurándose la posibilidad de realizar proyectos de mayor envergadura y complejidad organizativa, pudiendo ofrecer a los clientes soluciones de calidad.

Cada una de las áreas de proceso evaluadas exige el cumplimiento de un conjunto de prácticas específicas y genéricas definidas en el propio modelo. Para su alineación con el modelo, D2D ha modelado un sistema partiendo de la base de la herramienta de Gestión Redmine.

Los beneficios más importantes que aporta CMMI de nivel II son la institucionalización y estandarización de los procesos, la obtención de un absoluto control de la planificación y seguimiento de los proyectos, la implantación de una filosofía de mejora continua, el ahorro de coste y el aumento en la calidad del producto terminado. El objetivo consiste en lograr un mayor nivel de satisfacción de los clientes. Con este modelo se consigue un ahorro notable de tiempo y recursos, al mejorar la fiabilidad de la planificación y disminuir el retraso medio de las tareas. Así, se consigue reducir notablemente los trabajos derivados del mantenimiento posterior, aumentando la efectividad sobre la planificación realizada.

Con la implantación de CMMI, la firma obtiene una mejora notable en la calidad final de producto, al reducir el número de defectos derivados de inconsistencias con los requisitos. Por otro lado, la aplicación del modelo le permitirá detectar incidencias en las fases tempranas del ciclo de vida del proyecto con el importante ahorro de costes que supone. La obtención de esta certificación de calidad influirá positivamente en la organización gracias a la mayor capacidad de control, predicción y visibilidad en el desarrollo de proyectos de software.

A continuación se detallan las personas implicadas en el proyecto según sus roles:

- Cliente: D2D
- Usuario final: Luis Saragüeta (Responsable de calidad de D2D)
- Interlocutor: Andrés Escudero (Gerente de D2D)
- Desarrollador: David Ramírez

1.2.2 Situación actual

En el cliente se utiliza Redmine para gestionar los proyectos. Ésta es una aplicación web desarrollada en el *framework Ruby on Rails*, lo que le permite ser multi-plataforma y admitir varias bases de datos como MySQL, PostgreSQL o SQLite. La versión de la que hace uso el cliente se basa en la entrega 0.7.3 de Redmine.org.

Entre sus características y utilidades podemos destacar las siguientes:

- Visión general del proyecto así como su configuración.
- Se muestra el avance del proyecto y el % porcentaje que queda para terminar un hito.
- Peticiones: Son las unidades de trabajo; pueden ser tareas, errores, mejoras… etc. Estas se asignan a personas y se puede ir siguiendo su evolución, tiempo dedicado, comentarios… etc.
- Noticias del proyecto.
- Documentos.
- Wiki para crear la documentación necesaria en cada proyecto.
- Posibilidad de crear foros en cada proyecto.
- Repositorio: Podemos conectarlo a un repositorio de código que trabaje con GIT, SVN, CVS… etc.
- Control de accesos por roles.
- Se dispone de una página personal donde el usuario puede ver todo lo relacionado con sus tareas pendientes, las que él ha asignado, calendario… etc.
- Soporte multi-idioma.

1.2.3 Magnitudes o características principales que definen el PFC

Según la metodología del cliente, basada en el modelo CMMI nivel 2, a lo largo de todos los proyectos se distinguen diversas fases en las cuales se ejecutan distintas actividades. Independientemente del ciclo de vida del proyecto se distinguen el mismo número de fases en todos ellos: Inicio, Gestión, Análisis y Diseño, Desarrollo, Entrega, Soporte y Cierre.

La gestión de riesgos, la que compete al plug-in que se desea desarrollar para Redmine, comienza dentro de la fase de Inicio y continúa a lo largo de toda la vida del proyecto dentro de las tareas de Gestión. Con dicho plug-in se logrará, a través de un asistente, detectar e identificar posibles riesgos que puedan surgir a lo largo de todo el

ciclo de vida del proyecto así como proponer tanto acciones preventivas como correctivas para evitar y solucionar dichos riesgos.

Por parte del cliente se informa de que la aplicación Redmine corre sobre un servidor Windows 2008 Server Standard con acceso al Directorio Activo, al servidor corporativo de correo electrónico y a los repositorios de SVN. Se utiliza MySQL como motor de base de datos.

El *plugin* de riesgos deberá, además, ser totalmente configurable por parte del administrador de Redmine para que sea lo más versátil posible y que no esté completamente vinculado a los estándares definidos por este cliente. Por tanto, se deberá crear una parte de administración con la que poder introducir y configurar todos los parámetros que sean necesarios para el correcto funcionamiento de dicho *plugin*.

1.3 EQUIPOS UTILIZADOS PARA LA REALIZACIÓN DEL PFC

1.3.1 Equipos de desarrollo:

Fabricante: Dell
Procesador: Intel Core 2 CPU 6400 @ 2,13Ghz 2,13Ghz
Memoria RAM: 2 Gb
Sistema Operativo: Windows Vista 32 bits

1.3.2 Servidor de base de datos integración

Sistema Operativo: Microsoft Windows 2003 Server

1.3.3 Servidor de base de datos pre-producción

Sistema Operativo: Microsoft Windows 2003 Server

1.3.4 Servidor de base de datos producción

Sistema Operativo: Microsoft Windows 2003 Server

1.3.5 Servidor de producción

Sistema Operativo: Microsoft Windows 2008 Server Standard

1.4 ASPECTOS TEÓRICOS

1.4.1 Ruby:

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995.

Combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU.

Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Actualmente, a fecha de abril de 2011 la versión estable de Ruby es la 1.9.2.

1.4.1.1 Semántica

Ruby está orientado a objetos: todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas, (como enteros, booleanos, y "nil"). Las variables siempre son referencias a objetos, no los objetos mismos.

Ruby soporta herencia con enlace dinámico, mixins y patrones singleton (pertenecientes y definidos por una sola instancia más que definidos por la clase). A pesar de que Ruby no soporta herencia múltiple, las clases pueden importar módulos como mixins.

La sintaxis procedural está soportada, pero todos los métodos definidos fuera del ámbito de un objeto son realmente métodos de la clase Object. Como esta clase es padre de todas las demás, los cambios son visibles para todas las clases y objetos.
Ruby ha sido descrito como un lenguaje de programación multiparadigma: permite programación procedural (definiendo funciones y variables fuera de las clases haciéndolas parte del objeto raíz Object), con orientación a objetos, (todo es un objeto) o funcionalmente (tiene funciones anónimas, clausuras o closures, y continuations; todas las sentencias tiene valores, y las funciones devuelven la última evaluación).

Soporta introspección, reflexión y metaprogramación, además de soporte para hilos de ejecución gestionados por el intérprete. Ruby tiene tipificado dinámico, y soporta polimorfismo de tipos (permite tratar a subclases utilizando el interfaz de la clase padre). Ruby no requiere de polimorfismo de funciones (sobrecarga de funciones) al no ser fuertemente tipado (los parámetros pasados a un método pueden ser de distinta clase en cada llamada a dicho método).

1.4.2 *Ruby On Rails (Framework)*

Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).

Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de librerías y aplicaciones Ruby. Actualmente, a fecha de abril de 2011 la última versión publicada del framework es la 3.0.

1.4.2.1 Arquiectura MVC

Las piezas de la arquitectura Modelo Vista Controlador en Ruby on Rails son las siguientes:

- Modelo

En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos.
En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase Imagen tiene una definición has_many :comentarios, y existe una instancia de Imagen llamada a, entonces a.comentarios devolverá un array con todos los objetos Comentario cuya columna imagen_id (en la tabla comentarios) sea igual a a.id.

Las rutinas de validación de datos (p.e. validates_uniqueness_of:checksum) y las rutinas relacionadas con la actualización (p.e. after_destroy:borrar_archivo, before_update:actualizar_detalles) también se especifican e implementan en la clase del modelo.

- Vista

En MVC, Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador.

Con frecuencia en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML. El método que se emplea en Rails por defecto es usar Ruby Embebido (archivos.rhtml o archivos.html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP.
Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del controlador y los fragmentos pueden invocarse unos a otros.

- Controlador

En MVC, las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web.

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general en la forma http://aplicacion/ejemplo/metodo, que invoca a EjemploControllermetodo, y presenta los datos usando el archivo de plantilla /app/views/ejemplo/metodo.rhtml, a no ser que el método redirija a algún otro lugar.

Rails también proporciona andamiaje, que puede construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes.

1.4.3 MYSQL

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

MySQL AB - desde enero de 2008 una subsidiaria de Sun Microsystems - desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la GNU/GPL para cualquier uso compatible con esta licencia, pero las empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

Al contrario que proyectos como Apache, donde el software es desarrollado por una comunidad pública y el copyright del código está en poder del autor individual, MySQL es propiedad y está patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para

sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet

1.4.3.1 Historia

SQL (Lenguaje de Consulta Estructurado) fue comercializado por primera vez en 1981 por IBM, el cual fue presentado a ANSI y desde ese entonces ha sido considerado como un estándar para las bases de datos relacionales. Desde 1986, el estándar SQL ha aparecido en diferentes versiones como por ejemplo: SQL:92, SQL:99, SQL:2003.

MySQL es una idea originaria de la empresa opensource MySQL AB establecida inicialmente en Suecia en 1995 y cuyos fundadores son David Axmark, Allan Larsson, y Michael "Monty" Widenius. El objetivo que persigue esta empresa consiste en que MySQL cumpla el estándar SQL, pero sin sacrificar velocidad, fiabilidad o usabilidad.

Michael Widenius en la década de los 90 trató de usar mSQL para conectar las tablas usando rutinas de bajo nivel ISAM, sin embargo, mSQL no era rápido y flexible para sus necesidades. Esto lo conllevó a crear una API SQL denominada MySQL para bases de datos muy similar a la de mSQL pero más portable.

La procedencia del nombre de MySQL no es clara. Desde hace más de 10 años, las herramientas han mantenido el prefijo My. También, se cree que tiene relación con el nombre de la hija del cofundador Monty Widenius quien se llama My.

Por otro lado, el nombre del delfín de MySQL es Sakila y fue seleccionado por los fundadores de MySQL AB en el concurso “Name the Dolphin”. Este nombre fue enviado por Ambrose Twebaze, un desarrollador de Open source Africano, derivado del idioma SiSwate, el idioma local de Swazilandia y corresponde al nombre de una ciudad en Arusha, Tanzania, cerca de Uganda la ciudad origen de Ambrose.

1.4.3.2 Características distintas

Las siguientes características son implementadas únicamente por MySQL:

- Múltiples motores de almacenamiento (MyISAM, Merge, InnoDB, BDB, Memory/heap, MySQL Cluster, Federated, Archive, CSV, Blackhole y Example en 5.x), permitiendo al usuario escoger la que sea más adecuada para cada tabla de la base de datos.

- Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.

1.4.4 Aptana Studio (IDE)

Aptana es un entorno de desarrollo integrado (IDE) basado en un gran conocido para los desarrolladores Java como es el Eclipse, por supuesto también es un proyecto de código libre (open source), pero tiene una versión de pago con un par de características más.

Muy útil para los desarrolladores web ya que tiene la característica de autocompletar código, depurar en el navegador, viene listo para incluir en tus proyectos las principales librerías de ajax/javascript, te ayuda a validar xml, muestra la compatibilidad en diferentes exploradores y es muy interesante la posibilidad de agregar *plugin*s para realizar proyectos de Adobe Air y aplicaciones web para el iPhone. En el caso de éste como en otros IDE´s ó editores comentar en su contra la falta de resaltado de la sintaxis y autocompletado de javascript que lo tratan como texto plano.

EJECUCIÓN DEL PROYECTO (FASES)

2.1 Fase de inicio

En esta primera fase detallaremos la infraestructura del proyecto, los requisitos iniciales y que marcarán el desarrollo del *plugin* y analizaremos los posibles riesgos iniciales que pueden darse en el proyecto.

2.1.1 Creación de la infraestructura del proyecto

Se ha creado el repositorio del proyecto en la ubicación con permisos de lectura/escritura: **http://toledo:8080/svn/00031_Redmine** definidos para:

- Andrés Escudero
- Luis Saragüeta
- David Ramírez

2.1.2 Toma de requisitos iniciales

Tras la reunión con el cliente, éste comenta ciertos requisitos generales que serán aplicables a todo el conjunto del proyecto:

- La aplicación Redmine corre sobre un servidor Windows 2003 Server con acceso a toda la configuración.
- El cliente utiliza MySql como motor de BBDD.
- La aplicación estará disponible en inglés y castellano.
- El *plugin* mantendrá los estilos propios de Redmine.

Y más concretamente al desarrollo del *plugin*:

- Existirá una parte dedicada a la administración del propio *plugin* en la que únicamente el propio administrador de la aplicación podrá crear, editar o eliminar todos aquellos parámetros configurables del *plugin*.
- Se requiere la capacidad de poder identificar los diferentes riesgos que puedan surgir a lo largo de toda la vida del proyecto a través de un asistente. Dicho asistente, que se podrá utilizar en cualquier momento del proyecto y tantas

veces como se desee, consistirá en una batería de preguntas donde las respuestas se deberán de calificar como positivas o negativas según la configuración establecida por el administrador.

- Se requiere la capacidad de aplicar acciones preventivas y correctivas, las cuales deben llevar asociada la descripción de una tarea y poder asignar a estas uno o varios recursos de los que estén disponibles en la aplicación según la configuración del *plugin*.
- Se debe mantener un historial de los riesgos y de las acciones, tanto preventivas como correctivas, que se han dado en el proyecto así como su estado y duración en el tiempo.
- La utilización del *plugin*, excluyendo la parte destinada a su administración, queda restringida al jefe o jefes de proyecto del proyecto en cuestión y siguiendo las políticas de Redmine.

2.1.3 Análisis inicial de riesgos

En este apartado se detallan los posibles riesgos que pueden aparecer en el proyecto así como la probabilidad que existe de que el mismo riesgo pueda llegar a ser un problema para la organización, su impacto en la misma y su calificación final que será lo que motivará a seguir o no con dicho proyecto, la Tabla 1 recoge este análisis:

Riesgo	Condición de disparo	Prob.	Impacto	Cal.
Cliente desconocido	Se establece un nuevo interlocutor en el cliente o se trabaja con un cliente nuevo.	Muy alta	bajo	9%
Nuevas herramientas, entornos o metodologías	El cliente impone herramientas, entornos o metodología con los que no se ha trabajado (es previsible que supongan un coste extra en formación).	Muy alta	Moderado	18%

Equipo no experimentado con el cliente	El porcentaje de integrantes del equipo que han trabajado previamente con el cliente o sus interlocutores o que conocen sus metodologías, estructuras y procesos es inferior al 50%.	Muy alta	Bajo	9%
Equipo no experimentado en proyectos similares	El porcentaje de integrantes del equipo que han trabajado previamente en proyectos similares es inferior al 50%.	Muy alta	Bajo	9%
Falta de disponibilidad del equipo.	El equipo no se dedica exclusivamente al proyecto.	Alta	Bajo	7%

Tabla 1: Análisis de riesgos

2.2 Fase de análisis & diseño

En esta fase se mostrará el modelo relacional propuesto para el proyecto y se describirán detalladamente todas las tablas que lo conforman.

Además se analizarán todos los posibles casos de uso que existen tanto para la parte de administración como para la parte de usuario final.

2.2.1 Desarrollo del modelo de datos

El siguiente modelo de datos muestra las tablas de las que consta el *plugin*:

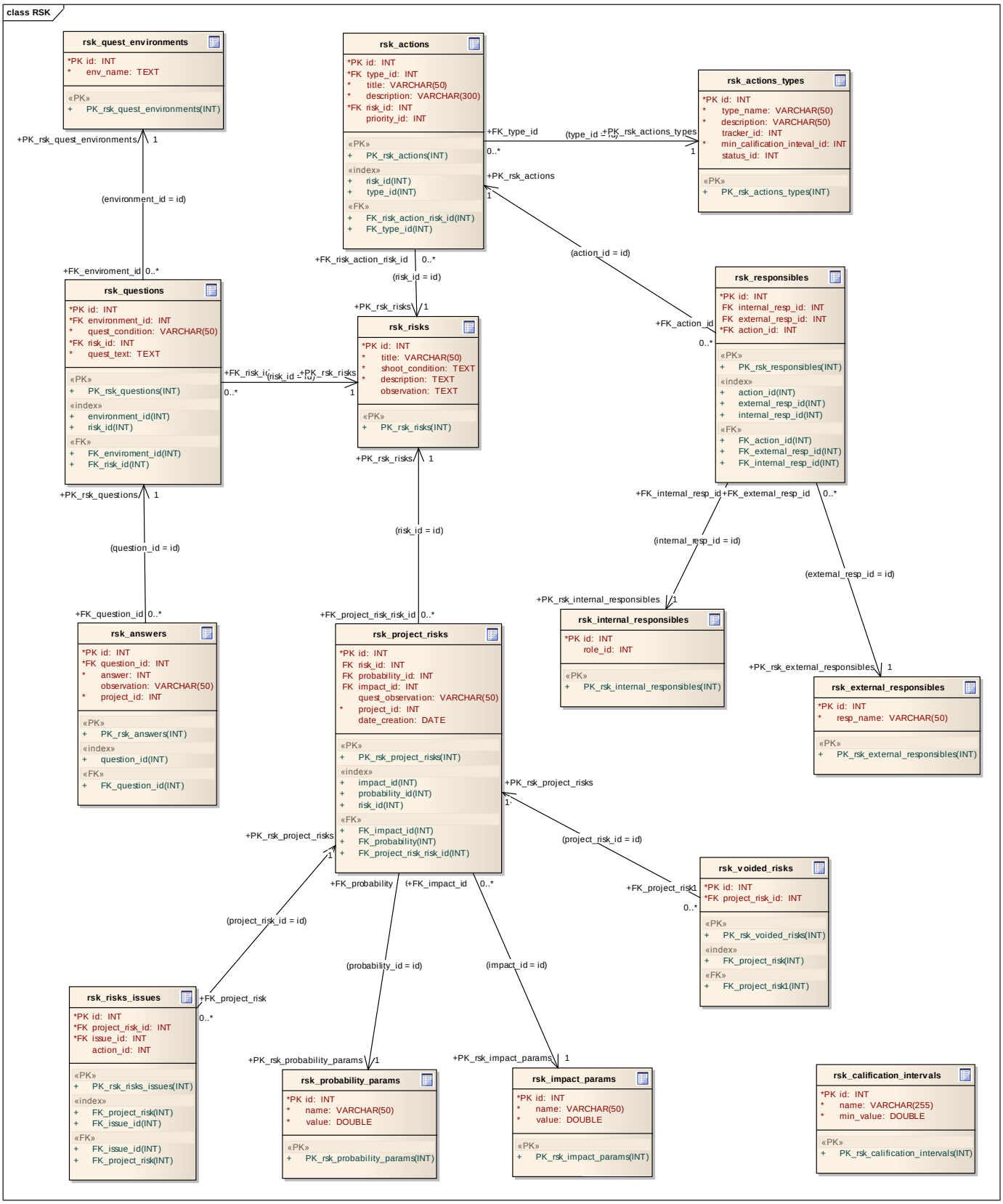
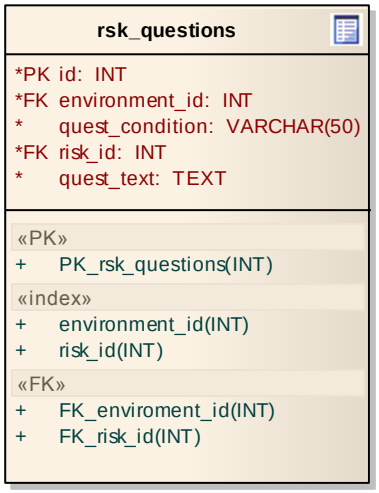
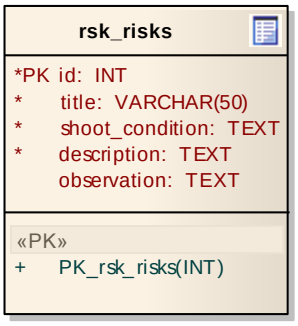
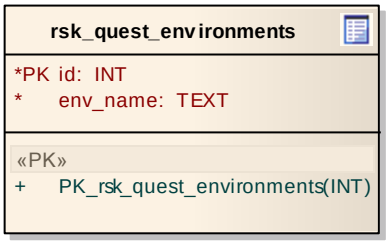
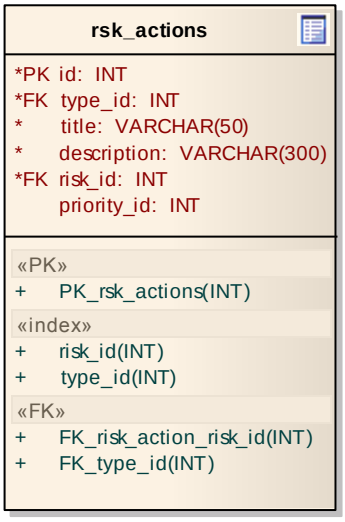
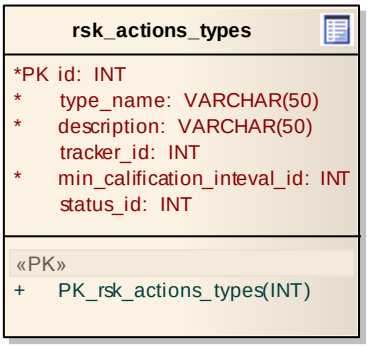


Figura 1: Modelo de datos

A continuación, se describe cada una de las tablas que conforman el modelo de datos. Haremos distinción entre tablas de administración (Tabla 2) y propias del uso del *plugin* (Tabla 3).

2.2.1.1 Tablas relacionadas con la administración del *plugin*

	Batería de preguntas de las que constará el asistente para la detección de los posibles riesgos.
	Catálogo de riesgos.

	Tabla que almacena los diferentes ámbitos en los que se suelen dar las preguntas del asistente: cliente, organización, equipo…
	Tabla que almacena las acciones que se pueden llevar a cabo tras la detección de un riesgo.
	Almacén con los diferentes tipos de acciones aplicables a cada riesgo.

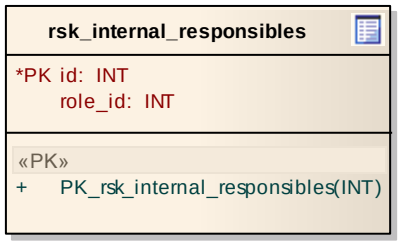
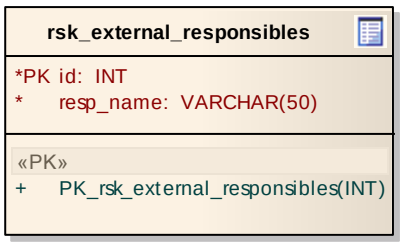
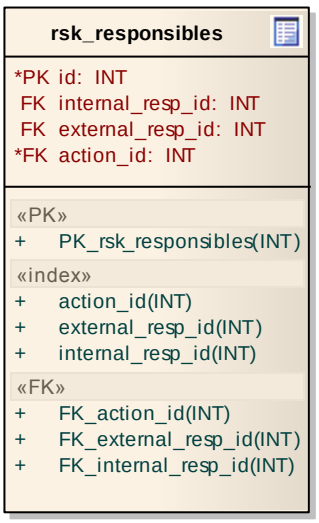
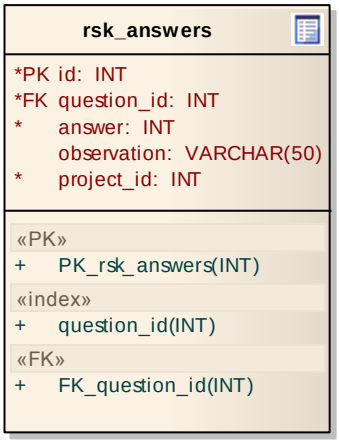
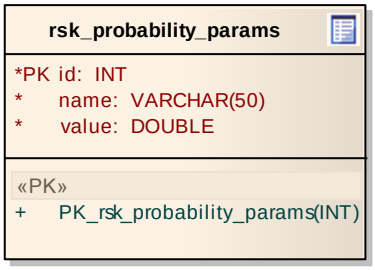
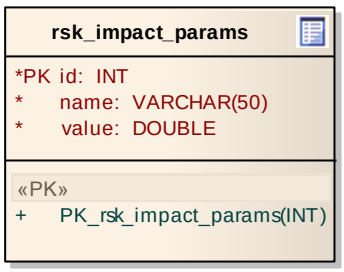
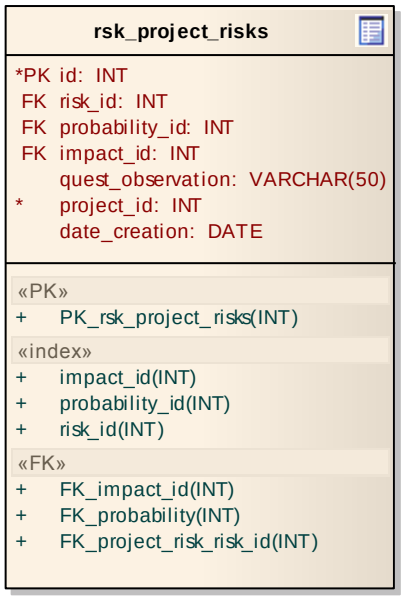
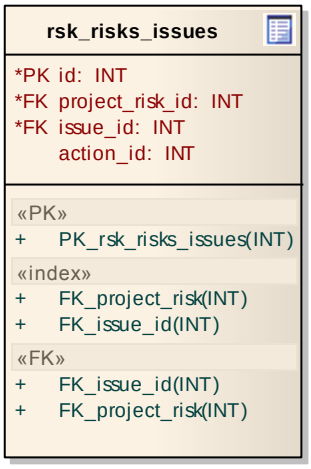
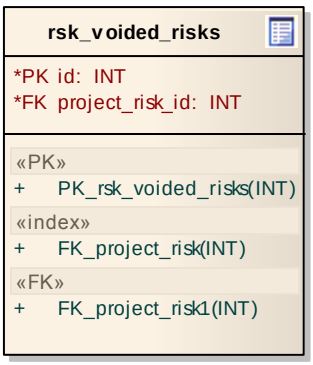
	Responsables internos de llevar a cabo cada una de las acciones.
	Responsables externos, a la organización, de llevar a cabo cada una de las acciones.
	Tabla que relaciona cada uno de los responsables, ya sean internos o externos, con las diferentes acciones que se pueden llevar a cabo.

Tabla 2: Diseño relacional a la administración del *plugin*

2.2.1.2 Tablas relacionadas con la ejecución del *plugin*

	Tabla donde se almacenan las respuestas a las preguntas del asistente.
	Parámetros de probabilidad.
	Parámetros de impacto.

	Tabla que almacena todos los riesgos del proyecto.
	Almacén de tareas con las acciones que se tienen que llevar a cabo trás la detección de un riesgo en el proyecto.
	Tabla que almacena los riesgos que se han eliminado en el proyecto.

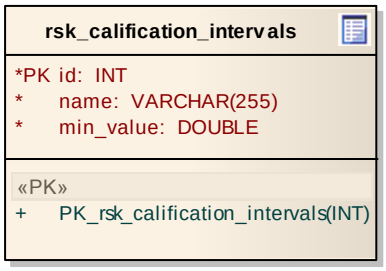
	Intervalos de calificación.

Tabla 3: Diseño relacional a la ejecución del *plugin*

2.2.2 Casos de uso

En ingeniería del software, un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

En nuestro *plugin* diferenciamos dos tipos de usuarios. El usuario administrador que se encargará de de instalar, configurar y mantener el *plugin* actualizado en todo momento (a través del panel de control desarrollado para tal uso) y el usuario final que será el 'actor' que propiamente utilizará el *plugin*.

A continuación se detallan estos diferentes escenarios así como las acciones que cada tipo de usuario puede realizar.

2.2.2.1 Escenarios del administrador

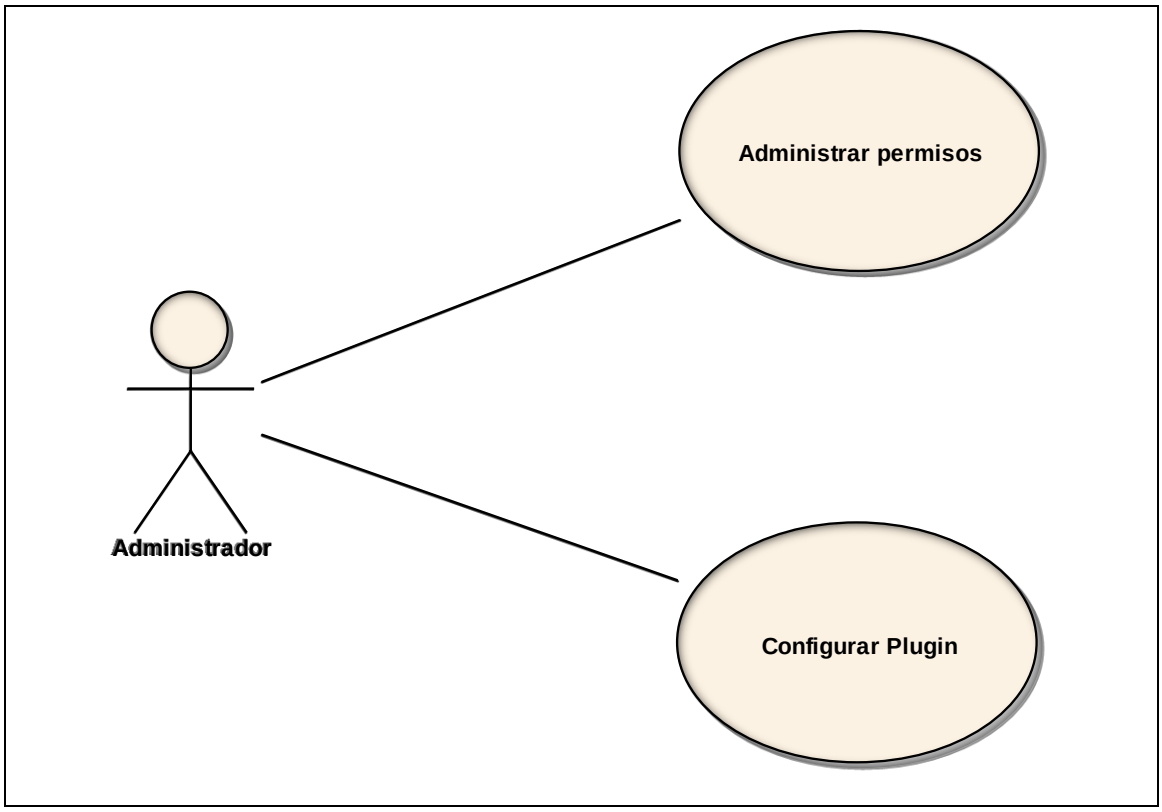


Figura 2: Iteración 0 (Administrador)

Administrar permisos	
Descripción	Se desea que el administrador de Redmine, una vez instalado el *plugin*, pueda conceder los permisos correspondientes para cada tipo de usuario.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Concesión de permisos guardada.
Flujo normal	1.El usuario administrador accede a la administración de permisos por el método definido para ello en Redmine. 2.Concederá permisos para cada rol. 3.Guardará los permisos

Configurar *plugin*	
Descripción	Se desea que el administrador de Redmine pueda configurar el *plugin* según los parámetros de su organización.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Configuración del *plugin* guardada.
Flujo normal	1.El usuario administrador accede a la configuración del *plugin* por el método definido para ello en Redmine. 2.Configurará el *plugin*. 3.Guardará los permisos

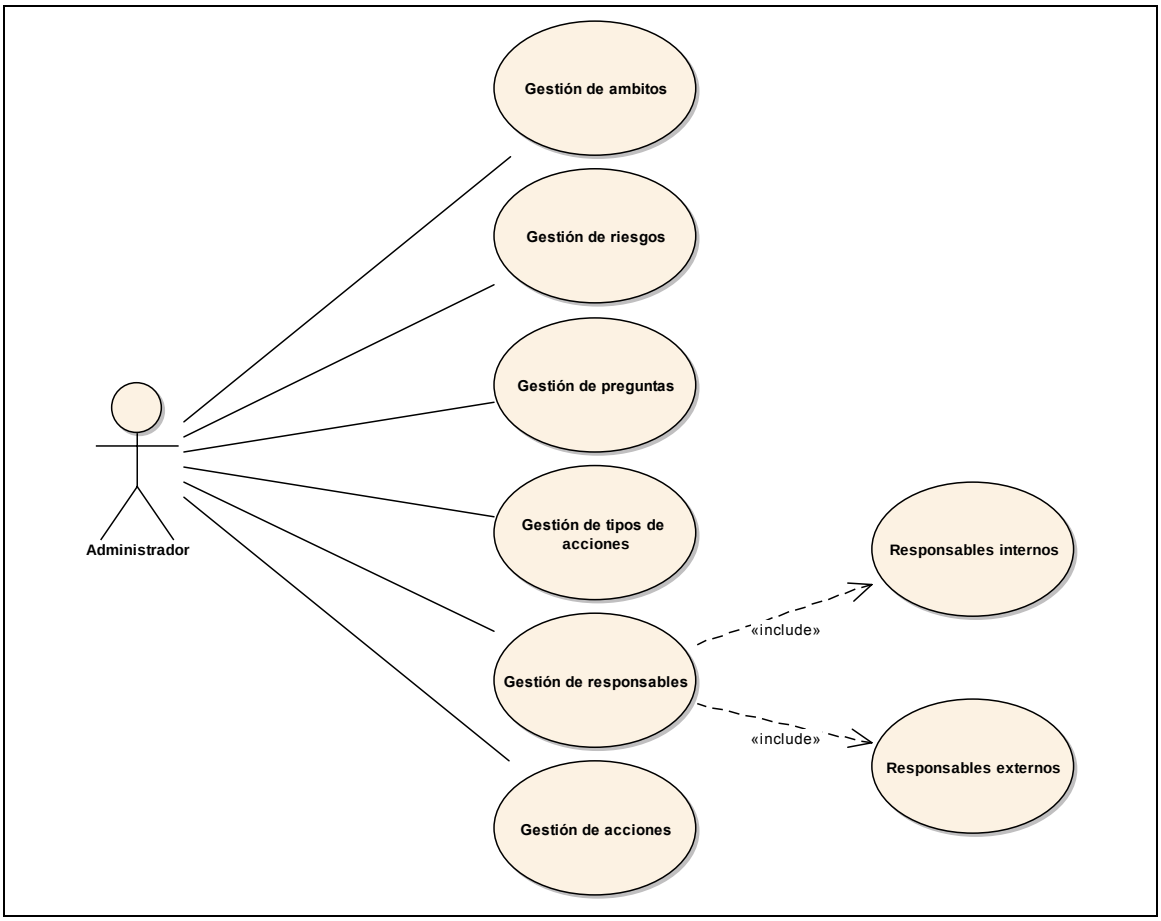


Figura 3: Iteración 1 (Administrador)

Gestión de ámbitos	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes tipos de ámbitos en los que los riesgos puedan aparecer.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Ámbitos creados, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona ámbitos en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de riesgos	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes riesgos que pueden aparecer en un proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Riesgos creados, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Riesgos' en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de preguntas	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes tipos de preguntas de las que consta el asistente.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Preguntas creadas, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Preguntas' en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de tipos de acciones	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes tipos de acciones.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Tipos de acciones creadas, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de responsables	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes responsables que gestionarán los riesgos de un proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsables creados, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables internos' o 'Responsables externos' en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de responsables internos	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes responsables internos (pertenecientes a la organización) que gestionarán los riesgos de un proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador. Existencia en Redmine de esos responsables.
Post-condición	Responsables internos creados, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables internos'. 3.El usuario realizará la operativa que necesite.

Gestión de responsables externos	
Descripción	Se desea que el administrador de Redmine pueda gestionar los diferentes responsables externos a la propia organización que gestionarán los riesgos del proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsables externos creados, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables externos' en el menú lateral. 3.El usuario realizará la operativa que necesite.

Gestión de acciones	
Descripción	Se desea que el administrador de Redmine pueda gestionar las diferentes acciones que realizarán los responsables para abordar los posibles riesgos que aparezcan.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Acciones creadas, editables y visibles.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Acciones' en el menú lateral. 3.El usuario realizará la operativa que necesite.

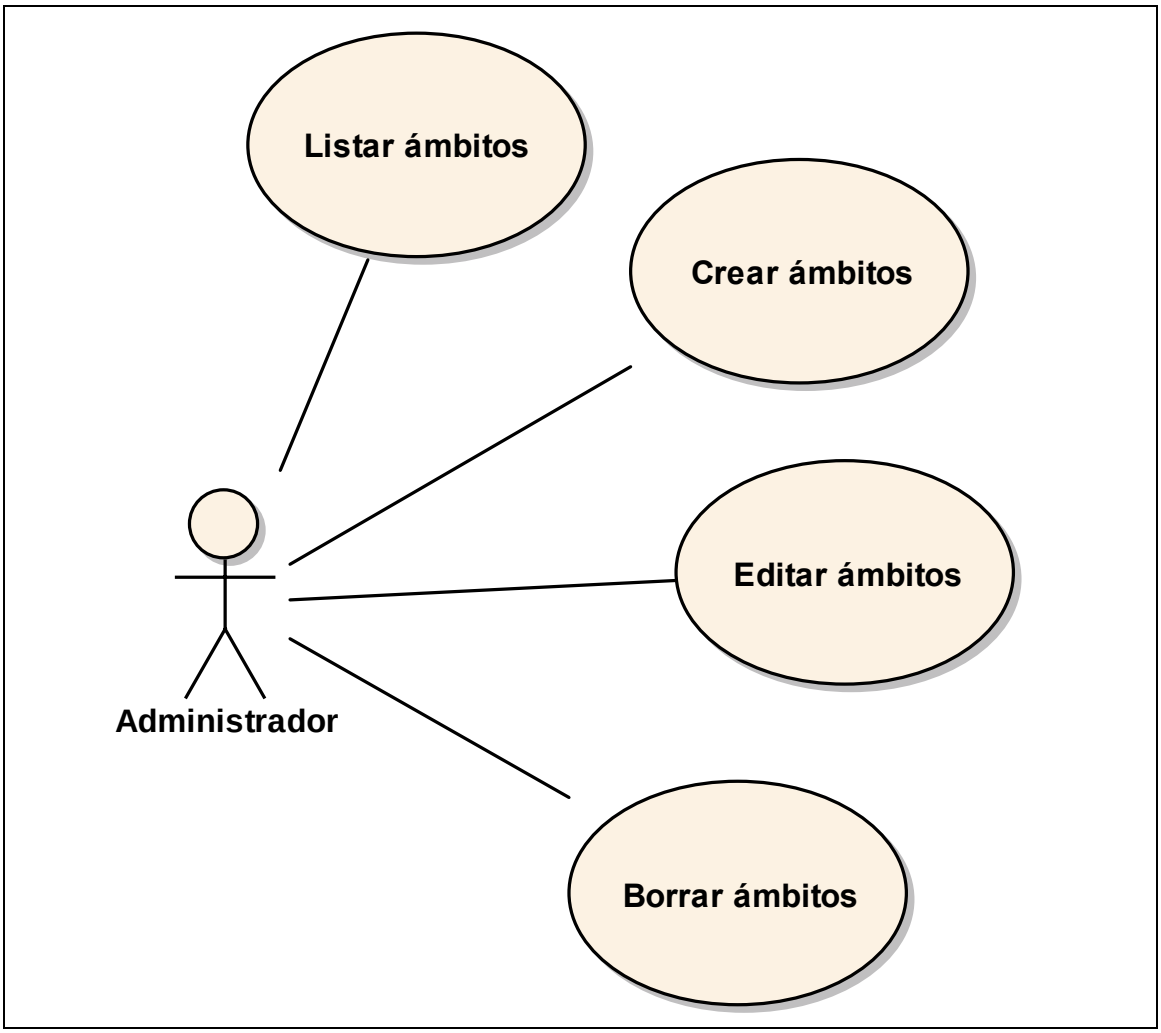


Figura 4: Iteración 2 – Gestión de ámbitos (Administrador)

Listar ámbitos	
Descripción	El usuario administrador tendrá acceso a un listado de los ámbitos definidos en la organización.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado de ámbitos ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Ámbitos' en el menú lateral.

Crear ámbitos	
Descripción	El usuario administrador creará los ámbitos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Ámbito creado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Ámbitos' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar ámbitos	
Descripción	El usuario administrador editará los ámbitos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Ámbito editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Ámbitos' en el menú lateral. 3. El usuario pulsa en el icono de edición de la pantalla del listado.

	4.El usuario modifica los datos del formulario y pulsa botón ‘Editar’. 5.El sistema le retorna al listado.

Borrar ámbitos	
Descripción	El usuario administrador borrará los ámbitos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Ámbito borrado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona ‘Ámbitos’ en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

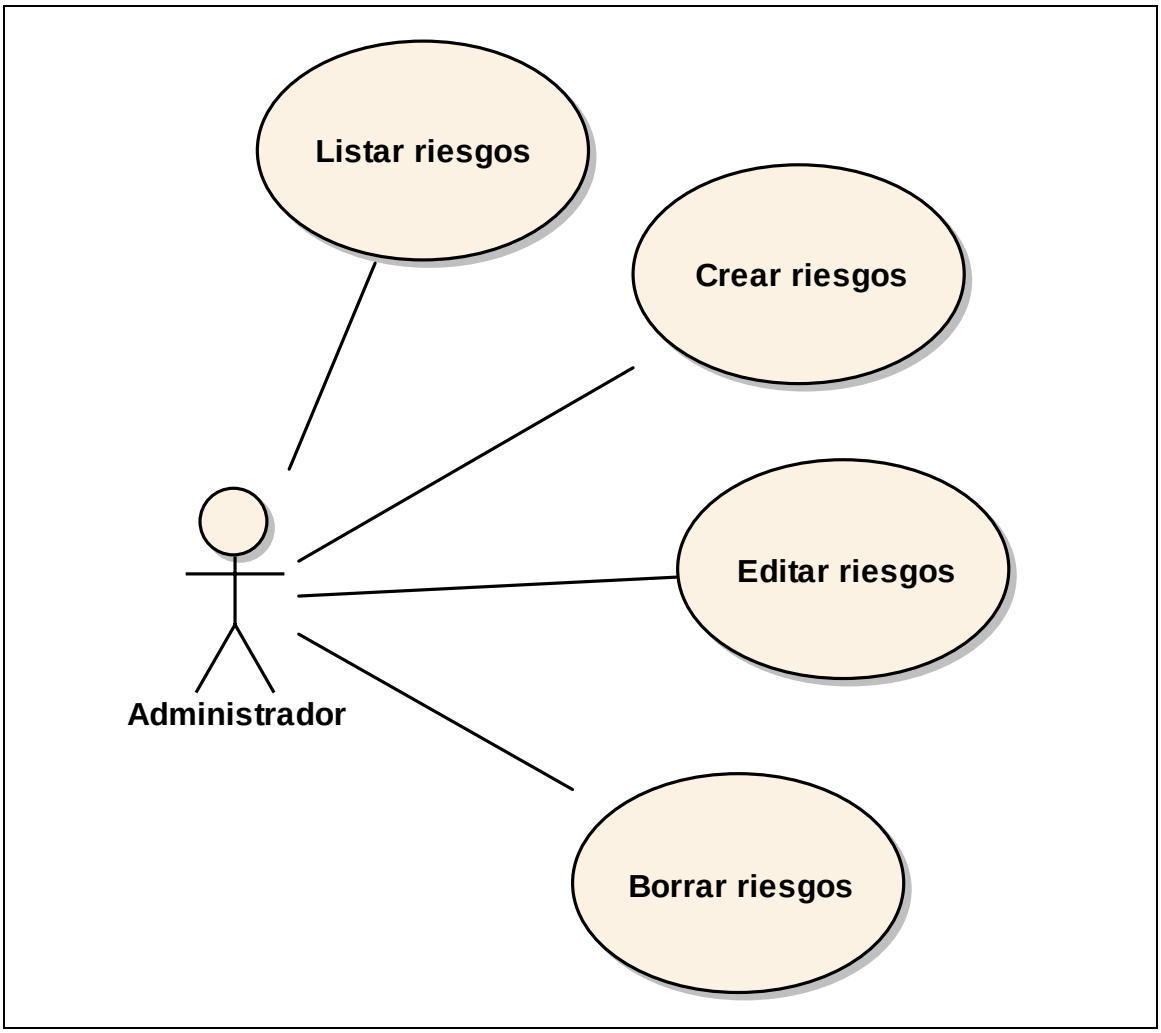


Figura 5: Iteración 2 – Gestión de riesgos (Administrador)

Listar riesgos	
Descripción	El usuario administrador tendrá acceso a un listado de los riesgos que se puedan dar.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado de riesgos ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Riesgos' en el menú lateral.

Crear riesgos	
Descripción	El usuario administrador creará los riesgos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Riesgo creado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Riesgos' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar riesgos	
Descripción	El usuario administrador editará los riesgos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Riesgo editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Riesgos' en el menú lateral. 3. El usuario pulsa en el icono de edición de la pantalla del listado.

	4.El usuario modifica los datos del formulario y pulsa botón ‘Editar’. 5.El sistema le retorna al listado.

Borrar riesgos	
Descripción	El usuario administrador borrará los riesgos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Riesgo borrado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona ‘Riesgos’ en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

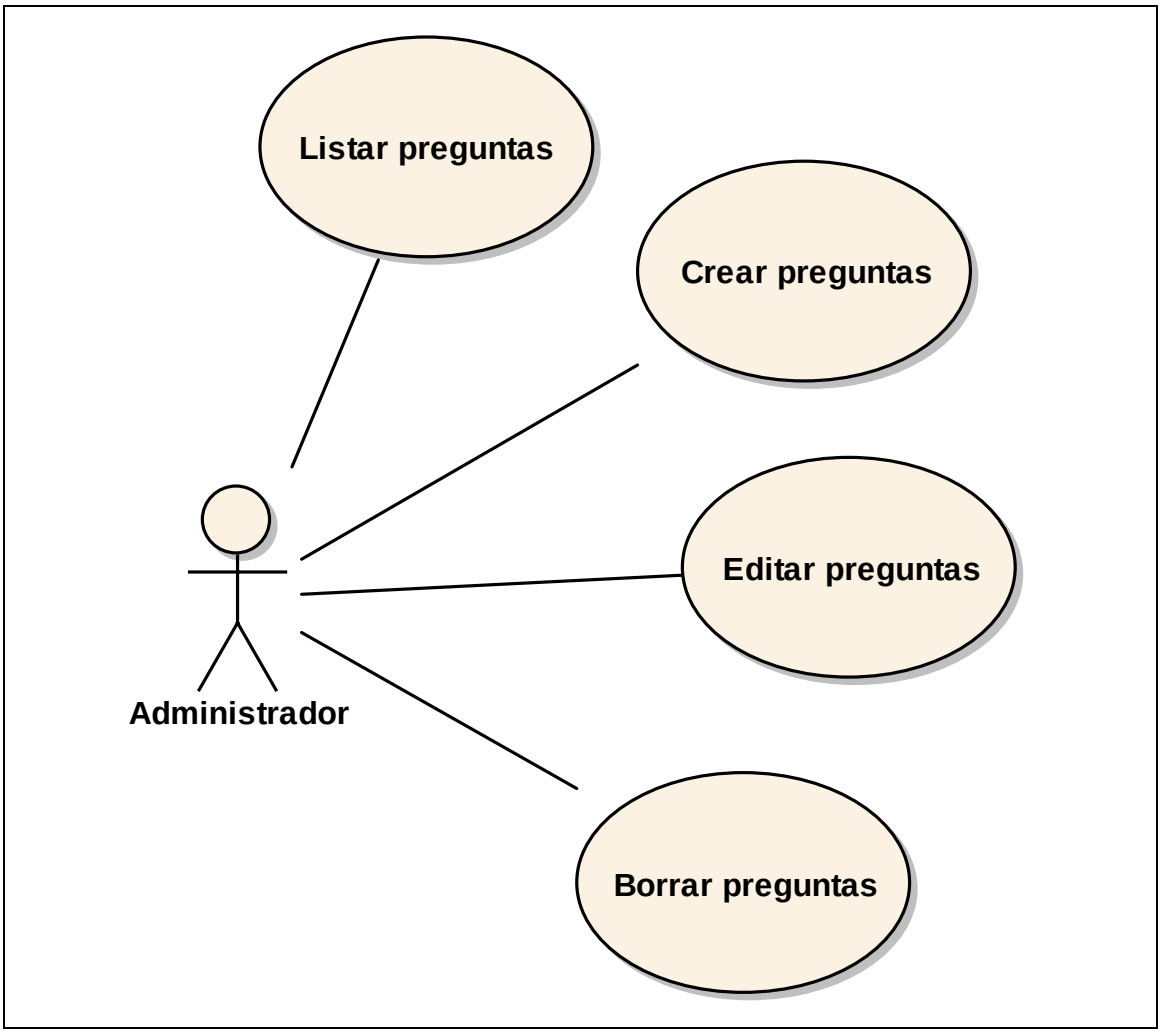


Figura 6: Iteración 2 – Gestión de preguntas (Administrador)

Listar preguntas	
Descripción	El usuario administrador tendrá acceso a un listado de las preguntas del asistente.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado de preguntas ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Preguntas' en el menú lateral.

Crear preguntas	
Descripción	El usuario administrador creará las preguntas que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Pregunta creada.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Preguntas' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar preguntas	
Descripción	El usuario administrador editará las preguntas que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Pregunta editada.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Preguntas' en el menú lateral. 3. El usuario pulsa en el icono de edición de la pantalla del listado.

	4.El usuario modifica los datos del formulario y pulsa botón ‘Editar’. 5.El sistema le retorna al listado.

Borrar preguntas	
Descripción	El usuario administrador borrará las preguntas que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Pregunta borrada.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona ‘Preguntas’ en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

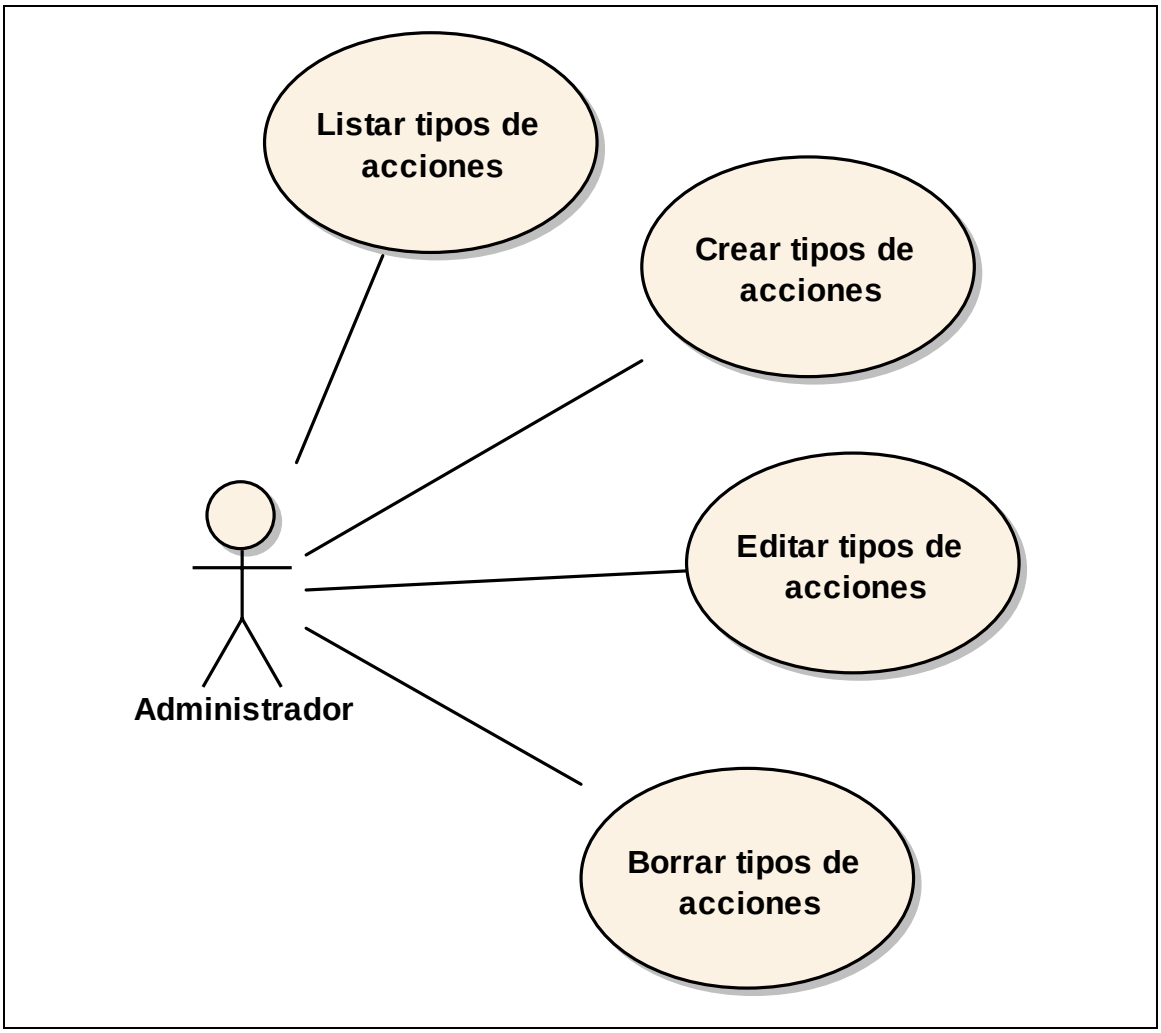


Figura 7: Iteración 2 – Gestión de tipos de acciones (Administrador)

Listar tipos de acciones	
Descripción	El usuario administrador tendrá acceso a un listado con los tipos de acciones que se hayan definido.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado de tipos de acciones ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral.

Crear tipos de acciones	
Descripción	El usuario administrador creará los tipos de acciones que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Tipo de acción creada.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar tipos de acciones	
Descripción	El usuario administrador editará los tipos de acción que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Tipo de acción editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral.

	3. El usuario pulsa en el icono de edición de la pantalla del listado. 4.El usuario modifica los datos del formulario y pulsa botón 'Editar'. 5.El sistema le retorna al listado.

Borrar tipos de acciones	
Descripción	El usuario administrador borrará el tipo deacción que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Tipo de acción borrado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

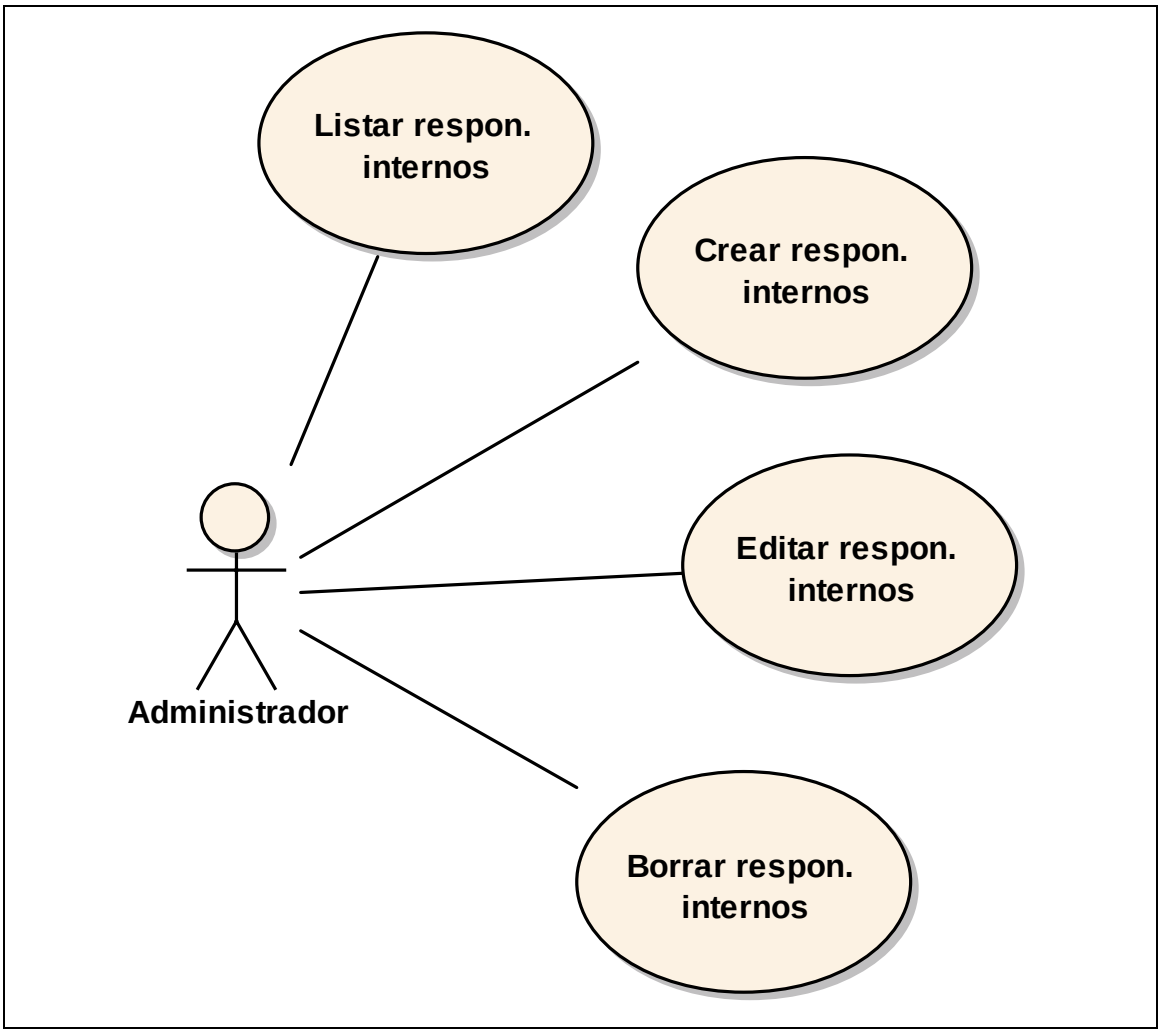


Figura 8: Iteración 2 – Gestión de responsables internos (Administrador)

Listar responsables internos	
Descripción	El usuario administrador tendrá acceso a un listado con los responsables internos que se hayan definido.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado con los responsables internos ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables internos '

Crear responsables internos	
Descripción	El usuario administrador creará los responsables internos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable interno creado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables internos' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar responsables internos	
Descripción	El usuario administrador editará los responsables internos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable interno editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Tipos de acciones' en el menú lateral.

	3. El usuario pulsa en el icono de edición de la pantalla del listado. 4.El usuario modifica los datos del formulario y pulsa botón 'Editar'. 5.El sistema le retorna al listado.

Borrar responsables internos	
Descripción	El usuario administrador borrará el responsable interno que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable interno borrado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables internos' en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

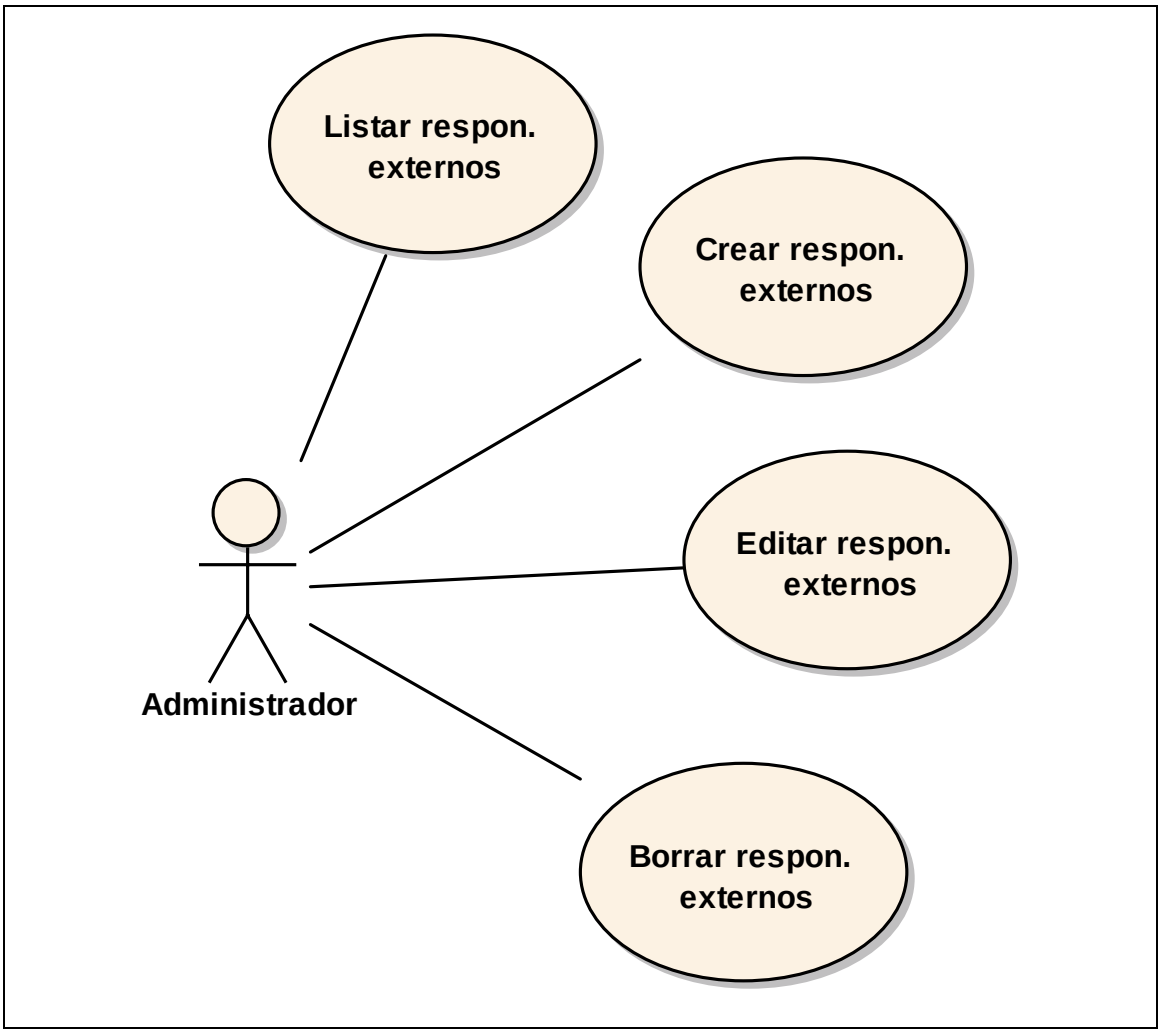


Figura 9: Iteración 2 – Gestión de responsables externos (Administrador)

Listar responsables externos	
Descripción	El usuario administrador tendrá acceso a un listado de los responsables internos que se hayan definido.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado con los responsables externos ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables externos '

Crear responsables externos	
Descripción	El usuario administrador creará los responsables externos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable interno creado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables externos' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar responsables externos	
Descripción	El usuario administrador editará los responsables externos que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable externo editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables externos' en el menú lateral.

	3. El usuario pulsa en el icono de edición de la pantalla del listado. 4.El usuario modifica los datos del formulario y pulsa botón 'Editar'. 5.El sistema le retorna al listado.

Borrar responsables externos	
Descripción	El usuario administrador borrará el responsable externo que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Responsable externo borrado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Responsables externos' en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

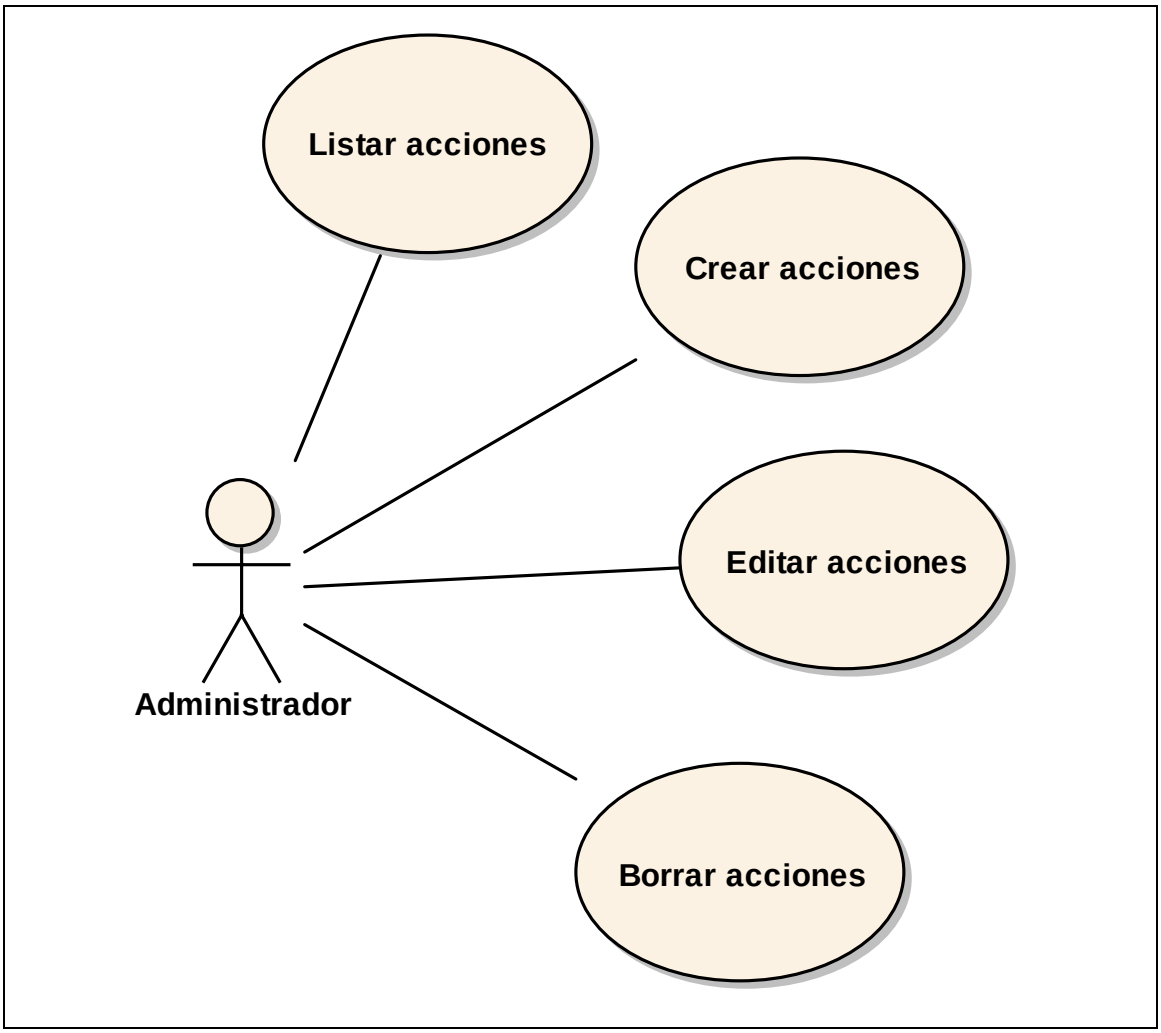


Figura 10: Iteración 2 – Gestión de acciones (Administrador)

Listar acciones	
Descripción	El usuario administrador tendrá acceso a un listado de las acciones que se hayan definido .
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Listado con las acciones ordenable y paginado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Acciones ' en el menú lateral.

Crear acciones	
Descripción	El usuario administrador creará las acciones que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador. Tipos de acciones, riesgos y responsables creados.
Post-condición	Responsable interno creado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Acciones' en el menú lateral. 3. El usuario pulsa en el enlace 'Nuevo' de la pantalla del listado. 4.El usuario rellena los datos del formulario y pulsa botón 'Crear'. 5.El sistema le retorna al listado

Editar acciones	
Descripción	El usuario administrador editará las acciones que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Tipo de acción editado.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona 'Acciones' en el menú lateral. 3. El usuario pulsa en el icono de edición de la pantalla del listado.

	4.El usuario modifica los datos del formulario y pulsa botón ‘Editar’. 5.El sistema le retorna al listado.

Borrar acciones	
Descripción	El usuario administrador borrará las acciones que necesite.
Pre-condición	Usuario correctamente autentificado en la aplicación con rol de administrador.
Post-condición	Acción borrada.
Flujo normal	1.El usuario accede a la configuración del *plugin*. 2.El usuario selecciona ‘Acciones’ en el menú lateral. 3. El usuario pulsa en el icono de borrado de la pantalla del listado. 4.El sistema le retorna al listado.

2.2.2.1 Escenarios del usuario

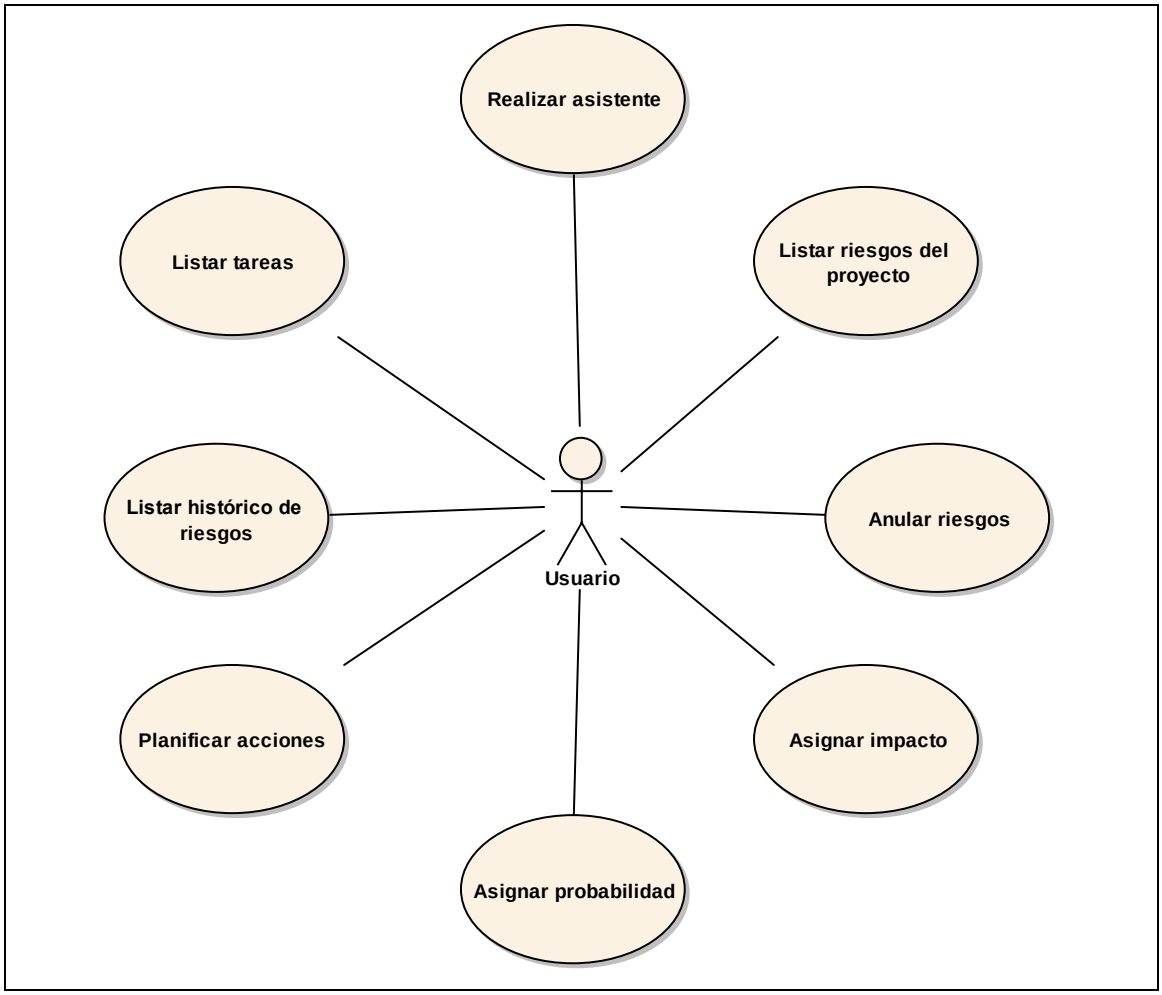


Figura 11: Iteración 0 – Usuario

Listar riesgos del proyecto	
Descripción	El usuario tendrá acceso a un listado de los riesgos activos que se generen a lo largo de la vida del proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado.
Post-condición	Listado con los riesgos ordenable y paginado.
Flujo normal	1.El usuario accede a través del menú de Redmine o a través de 'Riesgos activos' del menú lateral del *plugin*.

Realizar asistente	
Descripción	El usuario podrá realizar el asistente para la detección de los posibles riesgos a lo largo de todo el ciclo de vida del proyecto .
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado.
Post-condición	Riesgos del proyecto detectados.
Flujo normal	1.El usuario accede a través del menú de Redmine al *plugin*. 2.El usuario selecciona 'Asistente' en el menú lateral. 3.El usuario contesta a todas las preguntas del asistente. 4.El sistema le retorna al listado con los riesgos detectados.

Asignar probabilidad	
Descripción	El usuario asignará la probabilidad de que el riesgo se convierta en problema.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado.
Post-condición	Probabilidad asignada y calificación calculada.
Flujo normal	1.El usuario accede a través del menú de redmine o a través de 'Riesgos activos' del menú lateral del *plugin*. 2.El usuario cambia la probabilidad. 3.El usuario pulsa en 'actualizar'.

Asignar impacto	
Descripción	El usuario asignará el impacto que tiene el riesgo en el proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado.
Post-condición	Impacto asignado y calificación calculada.
Flujo normal	1.El usuario accede a través del menú de Redmine o a través de 'Riesgos activos' del menú lateral del *plugin*. 2.El usuario cambia el impacto. 3.El usuario pulsa en 'actualizar'.

Anular riesgo	
Descripción	El usuario anulará el riesgo.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado. No deben de existir tareas con acciones preventivas o correctivas en curso relacionadas con dicho riesgo.
Post-condición	Riesgo anulado.
Flujo normal	1.El usuario accede a través del menú de Redmine o a través de 'Riesgos activos' del menú lateral del *plugin*. 2.El usuario activa la anulación de los riesgos que estime oportunos. 3.El usuario pulsa en 'actualizar'.

Planificar acciones	
Descripción	El usuario podrá crear tareas para cada riesgo, siguiendo el modelo de Redmine, con las acciones correctivas y preventivas asociadas al mismo.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado. Deben existir riesgos activos en el proyecto y responsables asignados en la configuración del *plugin*.
Post-condición	Tareas asignadas y creadas.
Flujo normal	1.El usuario accede a través de 'Planificar acciones' en el listado principal de riesgos activos. 2.El usuario rellena el formulario de petición de tareas. 3.El usuario pulsa en 'Crear tareas'.

Listar tareas	
Descripción	El usuario tendrá acceso a un listado de las tareas con las acciones creadas para poder ver su estado.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado. Deben de existir tareas creadas.
Post-condición	Listado con las tareas ordenable y paginado.
Flujo normal	1.El usuario accede a través de 'Listado de tareas' del menú lateral del *plugin*.

Listar histórico de riesgos	
Descripción	El usuario tendrá acceso a un listado con los riesgos que han ido surgiendo a lo largo de la vida del proyecto.
Pre-condición	Usuario correctamente autentificado en la aplicación con permisos para acceder al *plugin*. *Plugin* configurado. Deben de existir o haber existido riesgos en el proyecto.
Post-condición	Listado con los riesgos ordenable y paginado.
Flujo normal	1.El usuario accede a través de 'Histórico de riesgos' del menú lateral del *plugin*.

INSTALACIÓN & FUNCIONAMIENTO

En este capítulo se detallarán aquellos aspectos importantes a tener en cuenta a la hora de instalar y configurar el *plugin* dentro de Redmine así como su arquitectura y correspondiente funcionamiento.

3.1 Arquitecturas

Previamente antes de explicar la configuración, es conveniente conocer la arquitectura propia de Redmine así como la del propio *plugin*.

3.1.1 Arquitectura de Redmine

En la siguiente figura se muestra la arquitectura (del directorio raiz) de Redmine, prestando especial atención al directorio "*vendor/plugins/*" donde se instalará nuestro *plugin*.

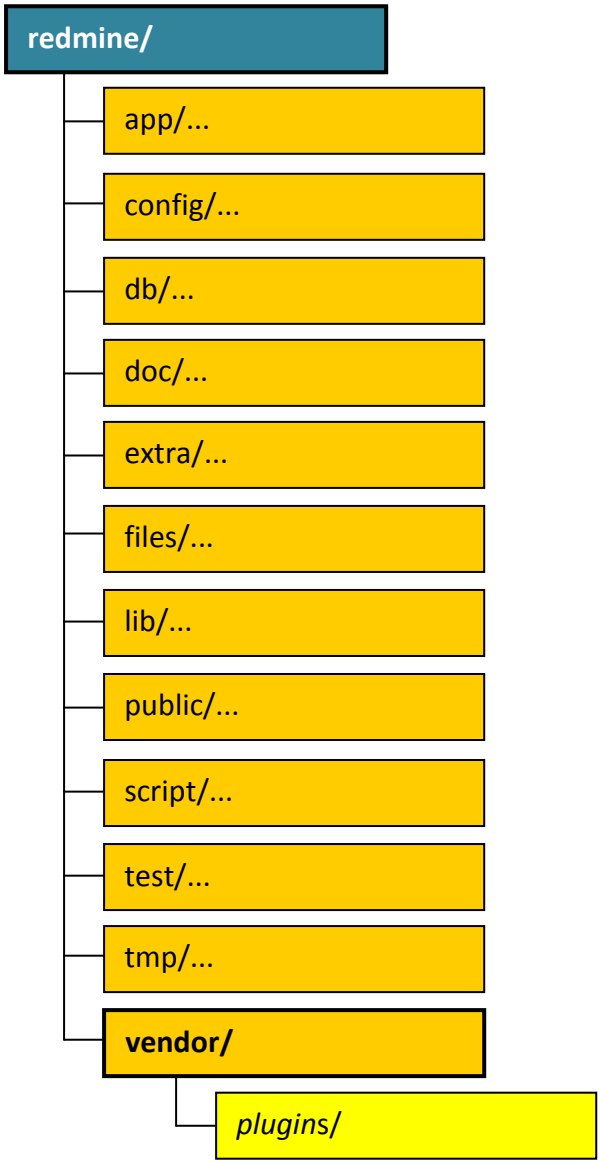


Figura 12: Arquitectura Redmine

3.1.2 Arquitectura del *plugin*

El *plugin* se instalará, como hemos explicado en el apartado anterior dentro del directorio "*plugins/*" y tendrá la siguiente arquitectura:

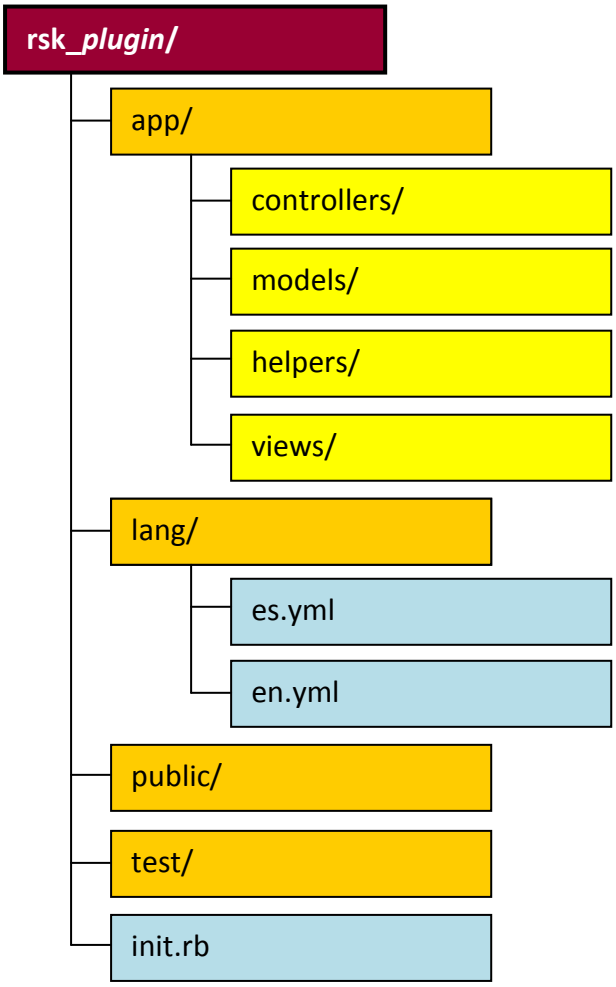


Figura 13: Arquitectura *plugin*

3.1.2.1 Descripción de la arquitectura

En los elementos de primer nivel encontramos:

- App: Código fuente de la aplicación.
- Lang: Ficheros de propiedades de los idiomas.
- Public: Imágenes y Hojas de estilo que se requieran a parte de las ya existentes en Redmine.
- Test: Herramientas para pruebas.
- Init.rb: Fichero de configuración del *plugin*.

En la carpeta app del *plugin* será donde alojaremos todo el código fuente de la aplicación. Debido a que es una aplicación Ruby on Rails, quedan perfectamente diferenciados los elementos en:

- Controllers: Controlador. Es el código fuente que realmente ejecuta las operativas. Recibe las peticiones, hace las operaciones oportunas y redirige a la Vista con el mismo nombre que la acción llamada, a otra vista o a otra acción del mismo o de otro controlador.
- Models: Modelo. Son las clases de acceso a datos. Definen las relaciones entre los objetos y las validaciones que se deben hacer.
- Helpers: Son clases con métodos de ayuda para las vistas. Es una forma de descargar las vistas de código.
- Views: Vistas. Una mezcla de código Ruby y html sin programación de lógica.

Debido a los requisitos tendremos dos ficheros de idioma:

- Inglés: en.yml
- Español: es.yml

3.2 Integración en Redmine

Podemos decir que integrar un *plugin* en Redmine es realmente sencillo y que lo único que tendremos que desarrollar será el fichero init.rb del que ya hemos hablado en el anterior apartado.

3.2.1 Fichero init.rb

```
require 'redmine'

RAILS_DEFAULT_LOGGER.info "Redmine RSK plugin started"
Redmine::Plugin.register :redmine_rsk do
 name 'Redmine Rsk plugin'
 author 'D2D'
 description 'This is a plugin for Redmine'
 version '0.0.1'

 project_module :redmine_rsk do
   permission :show_active_risks, {:rsk => [:index, :active_risks]},
:require => :member
   permission :show_historic, {:rsk => [:index, :historyrisks]},
:require => :member
   permission :edit_active_risks, {:rsk => [:index, :update]},
:require => :member
   permission :detect_risks, {:rsk => [:index, :assistant, :create,
:update]}, :require => :member
   permission :plan_actions, {:rsk => [:index, :createissue,
:actionissue, :issueslist]}, :require => :member
 end

 menu :project_menu, :rsk, { :controller => :rsk, :action => :index
}, :caption => :rsk_title, :after => :activity

 settings :default => {}, :partial => 'rsk/settings'

end
```

3.3.1.1 Descripción del fichero init.rb

- El *plugin* utiliza elementos de Redmine.
- En el log queremos que aparezca lo siguiente: ‘'Redmine Rsk *plugin*'.
- Definimos el creador, el nombre la descripción y la versión.
- Le decimos donde se encuentra la vista para la configuración del *plugin*.
- Damos un nombre al módulo.
- Creamos las instancias de los permisos para que desde Redmine pueden ser asignados a cada uno de los perfiles según proceda.
- Creamos un menú de proyecto diciendo a donde apunta (controller y action).

3.3 Configuración del plugin

El administrador de la aplicación será el encargado de configurar y administrar el *plugin*. En este apartado se explicará dicho proceso para su buen funcionamiento.

3.3.1 Concesión de permisos

En primer lugar, después de la activación del *plugin*, el administrador debe de otorgar permisos de acceso a los roles que la organización estime oportunos desde la pantalla de Redmine existente para ello. En la siguiente figura se muestran los roles que actualmente existen en Redmine para la organización y los correspondientes permisos para el *plugin* de riesgos.

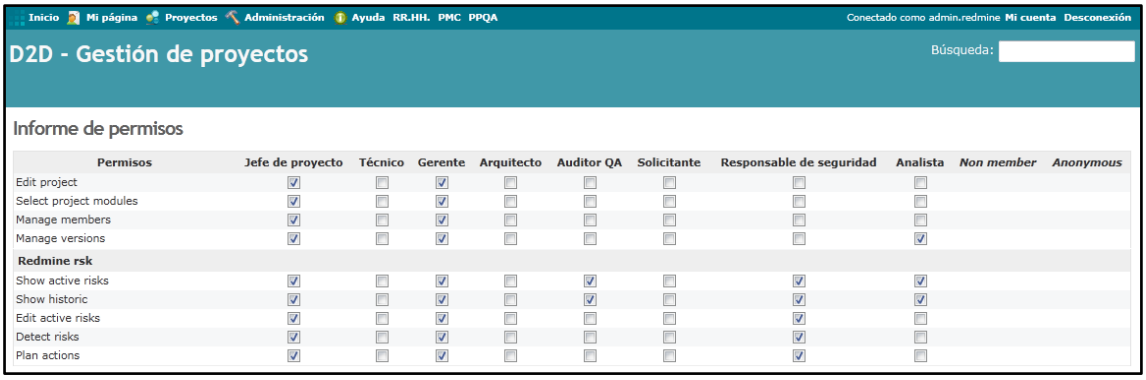


Figura 14 – Informe de permisos

En nuestro caso, en D2D, como se puede apreciar en la figura, únicamente los jefes de proyecto, gerentes y responsables de seguridad podrán acceder al *plugin*. El auditor QA sólo podrá ver que riesgos existen y el histórico de estos.

3.3.2 Gestión del *plugin*

El administrador será el encargado, como hemos comentado, de gestionar y administrar cada uno de los diferentes parámetros de los que consta el *plugin*, para lo cual dispondrá de una sección específica dentro de Redmine para facilitar su gestión.

Figura 15 – Gestión del RSK *plugin* (Listado de ámbitos)

Figura 16 – Gestión del RSK *plugin* (Parte del listado de riesgos)

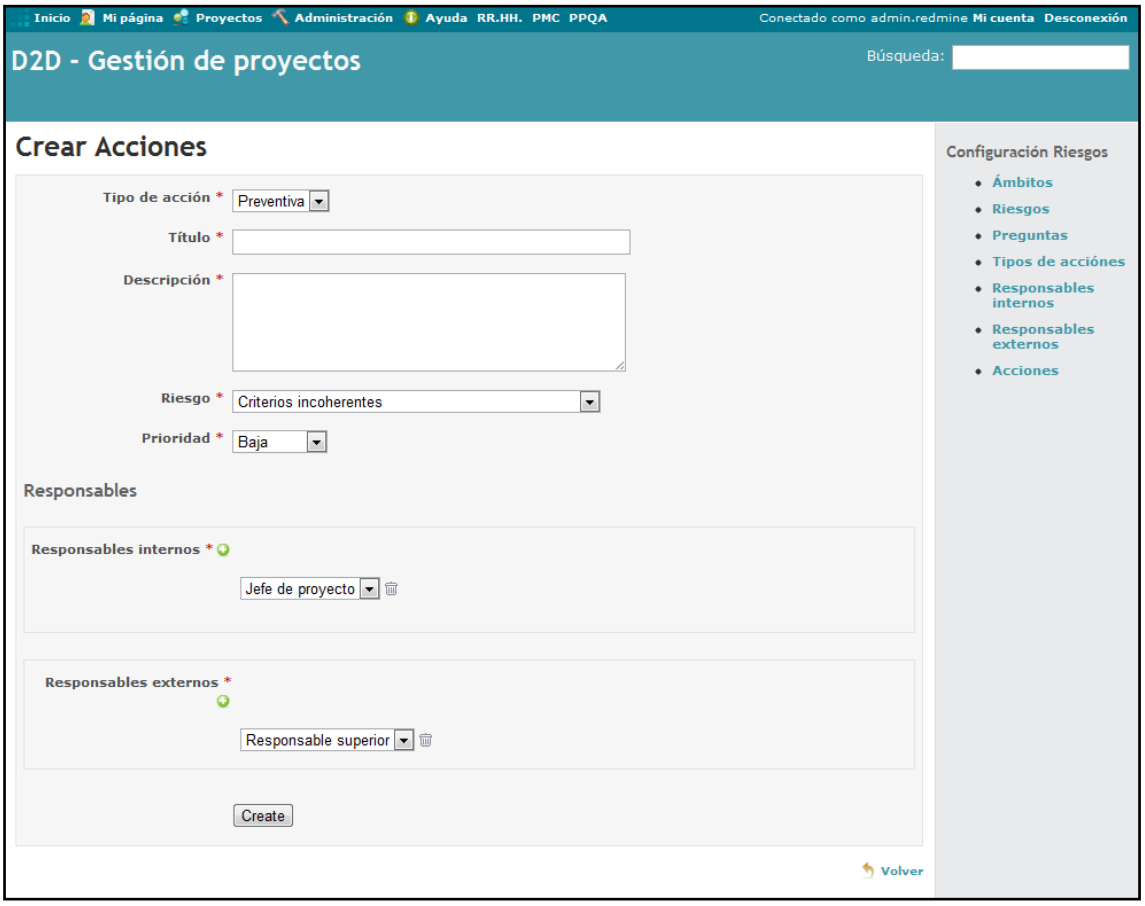


Figura 17 – Gestión del RSK *plugin* (Creación de acciones)

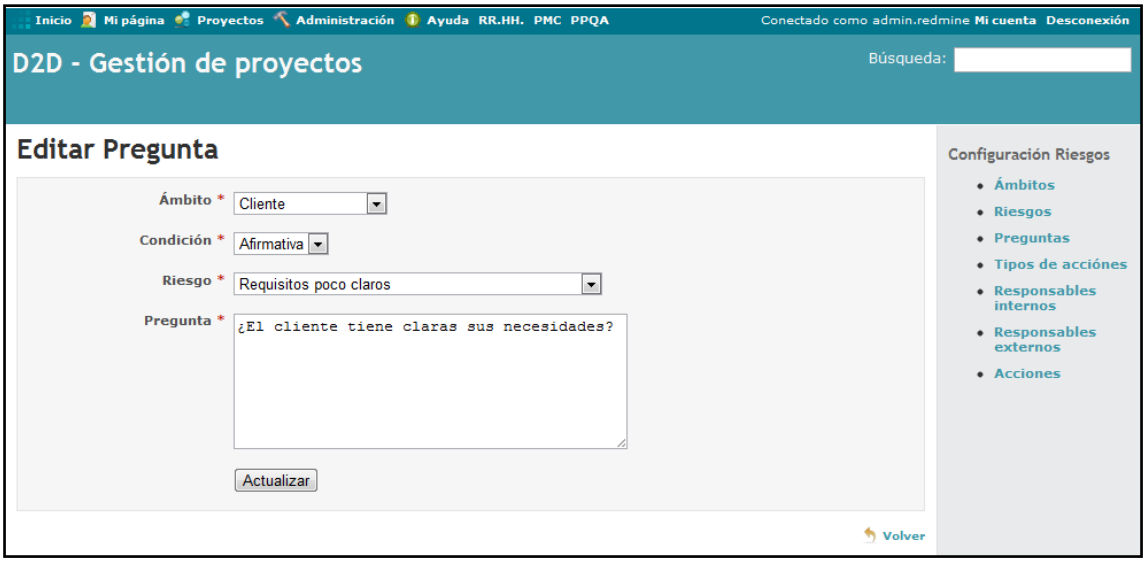


Figura 18 – Gestión del RSK *plugin* (Edición de preguntas)

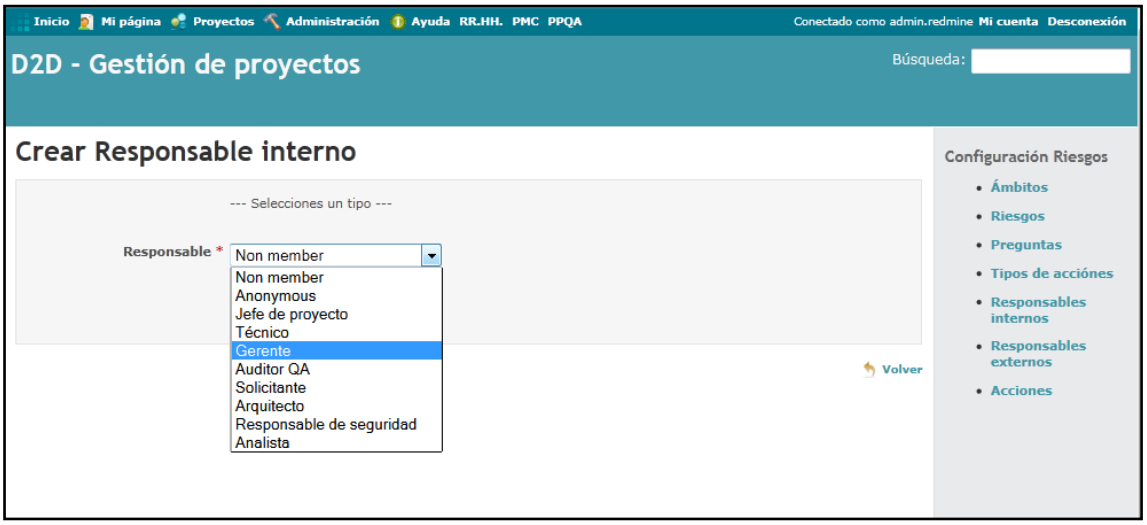


Figura 19 – Gestión del RSK *plugin* (Creación de responsables internos)

Tal y como se puede apreciar en las anteriores capturas de pantalla, el administrador podrá dar de alta nuevos parámetros, editarlos, eliminarlos y listarlos según se decida en la organización.

3.3.3 Internacionalización del *plugin*

Se utiliza la librería *GLoc* debido a que es la librería que utiliza Redmine. Nos obliga a tener una carpeta lang en el directorio principal del *plugin* y un fichero por cada idioma. En nuestro caso, en.yml para inglés y es.yml para español.

3.3.3.1 Caso de ejemplo

Para traducir el título de nuestro *plugin*, haremos lo siguiente:

1. En el fichero en.yml y en el fichero es.yml insertaremos

```
RSK_title: RSK
```

2. En la aplicación, en la *view* que corresponda nos bastará con pedir la internacionalización del identificador que hemos definido:

```
<%= l("RSK_title") %>
```

3.4 Funcionamiento del plugin

En este apartado se mostrará el funcionamiento básico que los usuarios del *plugin* harán para la correcta detección de riesgos y su correspondiente administración a través de acciones preventivas y correctivas.

3.4.1 Asistente

Según los requisitos iniciales, el *plugin* debía contar con un asistente para poder detectar los posibles riesgos de una manera sencilla. Las preguntas de dicho asistente, creadas, como hemos explicado en el apartado anterior, por el administrador, serán contestadas una a una por el usuario.

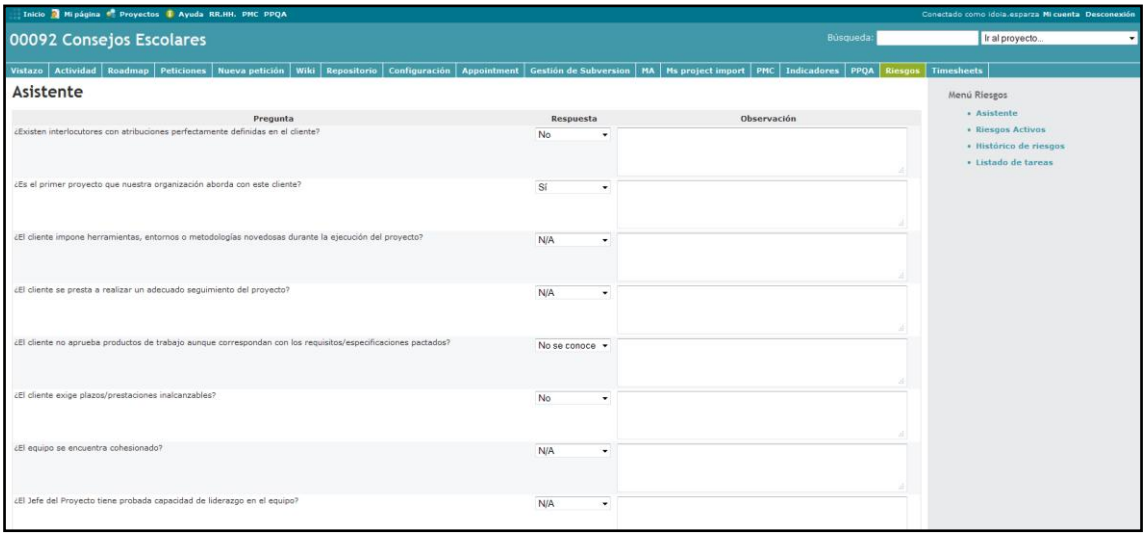


Figura 20 –RSK *plugin* (Asistente)

3.4.2 Listado de riesgos activos

Tras finalizar el asistente, el *plugin* devolverá una lista con los posibles riesgos que se puedan dar. El usuario deberá entonces asignar una probabilidad de que será aquella que estime oportuna a la hora de que el posible riesgo conlleve problemas y una probabilidad de impacto.

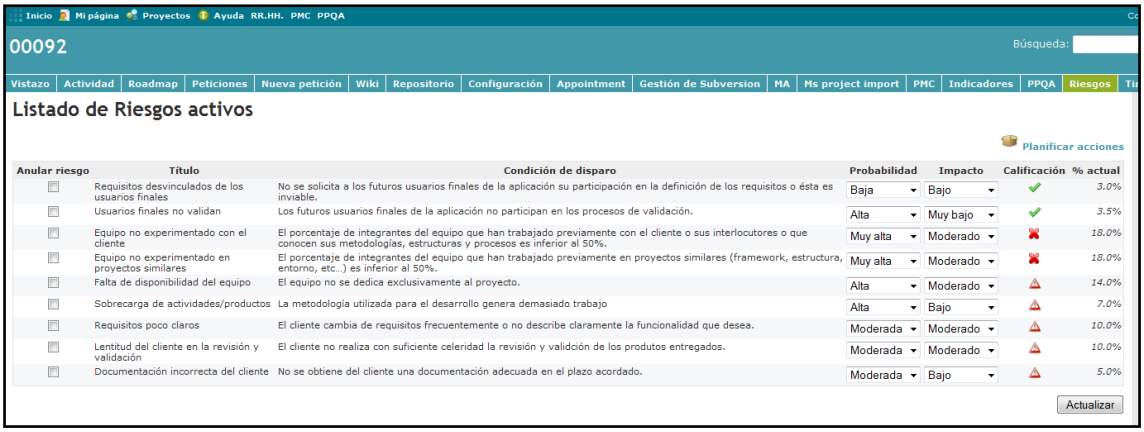


Figura 21 –RSK *plugin* (Listado de riesgos activos)

Una vez actualizados los riegos se calcula automáticamente su calificación y se muestra el resultado visualmente a través de varios iconos indicando la importancia de los mismos junto con su porcentaje real. Estas calificaciones serán al final en las que el usuario se fije para planificar las acciones oportunas de cara a prevenir o corregir los riesgos. Una vez realizadas estas tareas de prevención y corrección, los riesgos desaparecen o pueden ser anulados manualmente.

3.4.3 Histórico de riesgos

El asistente, como tal, puede ser contestado varias veces durante todo el ciclo de vida del proyecto generando posiblemente un cierto número de riesgos por lo que es muy interesante mantener un histórico de ellos así como del momento en el que se han producido y su calificación y estado actual.

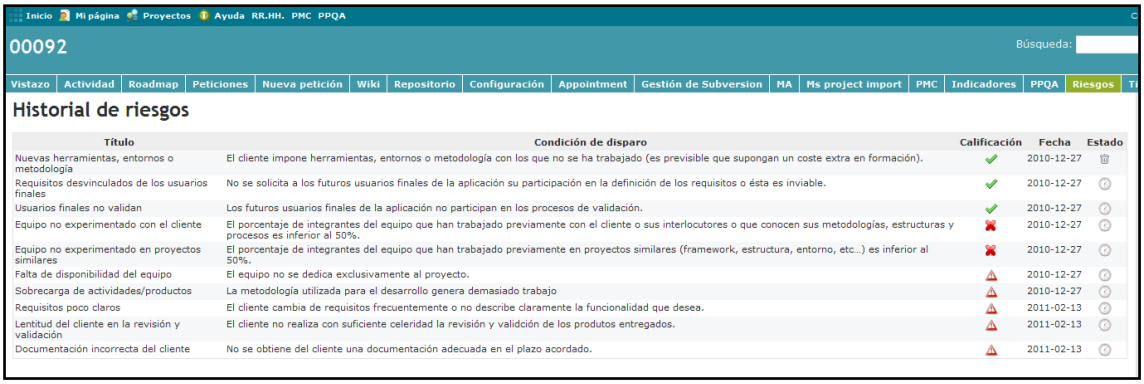


Figura 22 –RSK *plugin* (Histórico de riesgos del proyecto)

3.4.4 Planificando acciones

Como ya hemos comentado en varias ocasiones a lo largo de la memoria pueden existir varios tipos de acciones contra los riesgos según las preferencias de la organización.

Estas acciones realmente son tareas que se asignan a los diferentes responsables que el administrador a configurado. Para poder crear estas tareas los usuarios del *plugin* disponen de una pantalla en la que pueden crearlas como si de una tarea de Redmine cualquiera se tratara.

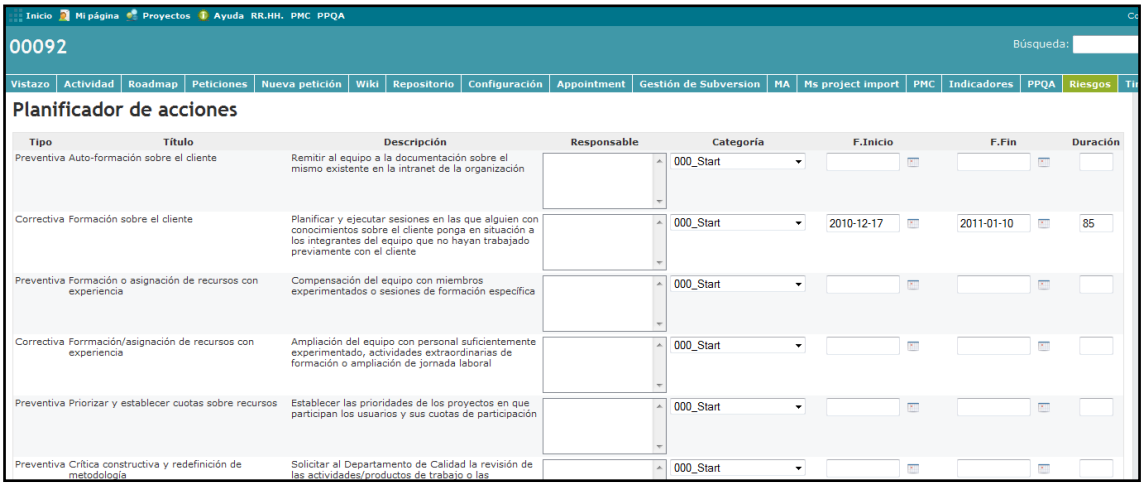


Figura 23 –RSK *plugin* (Planificador de acciones)

Como se aprecia en la captura de pantalla los usuarios deben de asignar o no un responsable a la acción, la cual, se encuentra acompañada de su tipo y su correspondiente descripción, una categoría, fechas de inicio y fin y una posible duración.

3.4.5 Listado de tareas

Por último, es interesante, también destacar la importancia de visualizar en un listado las tareas (con las acciones correspondientes y responsables asociados) agrupadas según su tipo que se han planificado así como su evolución y estado.

3.5 Pruebas

Para explicar las pruebas realizadas en el *plugin* previamente antes de su entrega podemos dividirlas entre pruebas unitarias y pruebas de carga.

3.5.1 Pruebas unitarias

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

En nuestro caso primero se desarrolló y probó el modulo de administración y gestion de riesgos con resultados satisfactorios y posteriormente con el modulo de usuario, propiamente el *plugin* obteniendo también buenos resultados.

3.5.2 Pruebas de carga

Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada.

En nuestro caso este número de peticiones es bastante bajo, se espera que no más de tres o cuatro usuarios estén utilizando el *plugin* en el mismo instante debido al tamaño de la organización. Habría que destacar que, realmente, el que lleva el peso de estas pruebas es Redmine por lo que este tipo de pruebas ya están demostradas.

CONCLUSIONES

4.1 Conclusiones a nivel técnico

Con el desarrollo del proyecto, las conclusiones a nivel técnico se centran sobre todo entorno al lenguaje Ruby y su *framework Rails*.

Nunca antes había programado en dicho lenguaje y la experiencia obtenida después de desarrollar el *plugin* es realmente buena. He podido descubrir la facilidad y rapidez de programar en *Ruby on Rails* obteniendo a su vez grandes resultados con tan solo unas pocas líneas de código

Me ha sorprendido gratamente, también, la aplicación Redmine a la hora de gestionar los proyectos así como su facilidad para incorporar nuevos *plugin*s que satisfagan las necesidades de cada organización como es el caso de D2D y la posibilidad de compartir los *plugin*s desarrollados.

4.2 Conclusiones del producto obtenido

Considero que con el desarrollo de este proyecto se ha conseguido, a través de la integración en Redmine, y con una amigable y sencilla interfaz, facilitar el trabajo de los jefes de proyecto de D2D a la hora de poder prever los posibles riesgos que se puedan dar a lo largo de todo el proyecto planificando a su vez las correspondientes acciones, tanto correctivas como preventivas, ahorrando tiempo y por lo tanto dinero.

A pesar de que la parte de administración del *plugin* puede requerir un gran esfuerzo de configuración por parte del administrador del sistema, posteriormente los beneficios son múltiples a la hora de organizar un proyecto.

4.3 Líneas futuras

A lo largo de este proyecto se ha desarrollado un producto bastante completo y que permite realizar diversas funcionalidades.

Como hemos comentado en el primer apartado de esta memoria, D2D sigue el modelo de gestión CMMI de nivel 2. Si en un futuro se obtuviese el nivel 3, habría que analizar en que aspectos se diferencia la gestión de riesgos y aplicarla al propio *plugin*.

VALORACIÓN PERSONAL

La valoración que hago del proyecto es totalmente positiva.

Al comenzar no tenía ningún tipo de experiencia y el desarrollar en un lenguaje y entorno, prácticamente desconocidos para mi, se hizo costoso. Nadie dijo que los comienzos son fáciles, pero conforme iba adaptándome a la metodología de trabajo y al nuevo lenguaje fui adquiriendo una gran cantidad de conocimientos que hicieron mucho más simple el desarrollo del proyecto tales como:

- Nuevos conocimientos en la gestión de proyectos. (Ciclo de vida de los proyectos desde que la organización se plantea su desarrollo hasta la fase de entrega del mismo).

- Conocimiento de una herramienta de gestión de proyectos como es Redmine tanto a nivel de usuario como de desarrollador.

- Conocimiento de un nuevo lenguaje y un nuevo *framework* como es *Ruby on Rails*.

- Conocimiento de un nuevo IDE como es Aptana, basado en el conocido Eclipse.

- Conocimientos propios a la metodología de trabajar en la propia organización, en este caso D2D.

Destacar también la satisfacción por ver como el proyecto actualmente cumple su propósito dentro de la propia empresa donde realmente se le da utilidad y pensar que todo el esfuerzo ha merecido la pena.

Por último me gustaría mostrar a D2D mi agradecimiento por brindarme la posibilidad de realizar con ellos este proyecto y darme la oportunidad de continuar actualmente con ellos y crecer como profesional.

BIBLIOGRAFÍA

El presente documento y el desarrollo del mismo se han basado en información obtenida de:

1. Página oficial de Redmine

   http://www.Redmine.org/

2. Tutorial para el desarrollo de *plugin*s para Redmine

   http://www.Redmine.org/wiki/Redmine/*Plugin*_Tutorial

3. Instalación Redmine

   http://www.Redmine.org/wiki/Redmine/RedmineInstall

4. Instalación de *plugin*s en Redmine

   http://www.Redmine.org/wiki/Redmine/*Plugin*s

5. Página oficial de Ruby en español

   http://www.Ruby-lang.org/es/

6. Página oficial de Ruby on Rails en español

   http://www.RubyonRails.org.es/

7. API Ruby on Rails

   http://api.RubyonRails.org/

8. Página oficial *Engines*

   http://Rails-engines.org/

9. Página oficial de Aptana Studio

   http://aptana.com/

10. Página Oficial MySql

    http://www.mysql.com/

11. Página Oficial CMMI

    http://www.sei.cmu.edu/cmmi/

12. Página oficial D2D

http://d2d.es/

13. Wikipedia

http://es.wikipedia.org/